DB2. Information Management Software

IBM

DB2 UDB ESE V8 non-DPF Performance Guide for High Performance OLTP and BI

Overview of DB2 UDB ESE V8 non-DPF architecture

Best practices for optimal performance

Problem determination scenarios

Nagraj Alur
Peter Farrell
Philip Gunning
Saeid Mohseni
Swaminaathan Rajagopalan

Redbooks

ibm.com/redbooks

International Technical Support Organization

DB2 UDB ESE V8 non-DPF Performance Guide for High Performance OLTP and BI

April 2004

SG24-6432-00

Note: Before using this information and the product it supports, read the information in "Notices" on page xix.

First Edition (April 2004)

This edition applies to Version 8, Release 1 DB2 Universal Database Enterprise Server Edition (ESE) (product number 5765-F41).

Contents

Figures

Tables

Examples

Notices

Preface

This IBM® Redbook will help you to develop, monitor, and tune DB2® UDB Version 8.1 non-database partitioning feature (non-DPF) Online Transaction Processing (OLTP) and Business Intelligence (BI) applications in the AIX® and Windows® environments.

Attention: The scope of this book is a DB2 UDB Version 8.1 non-DPF environment for the AIX and Windows platforms, and does not include a discussion of federated database functionality. It is aimed at a target audience of *experienced* DB2 application developers and database administrators (DBAs).

This book is organized as follows:

- **Chapter 1** provides a general overview of performance management concepts, and describes the high-level tasks that DBAs typically perform to ensure that their DB2 environment is performing adequately.
- **Chapter 2** provides a general overview of the architecture of DB2 UDB V8 from a performance management perspective. It also describes the flow of a transaction/query as it interacts with various processes and resources in DB2, and identifies potential performance bottlenecks along the way—both in a single user environment with no contention, and in a multi-user environment involving contention for various resources.
- **Chapter 3** describes the key performance drivers that impact OLTP and BI performance, and suggest best practices for achieving superior DB2 OLTP and BI performance. Application design and system performance considerations are discussed.
- **Chapter 4** describes key performance considerations associated with executing DB2 commands and utilities, and suggests best practices for achieving superior performance.
- **Chapter 5** provides an overview of AIX and Windows systems performance considerations that impact DB2 performance, and includes a discussion of operating system, memory, and disk considerations and recommendations. It also describes some of the performance monitoring and management tools available.
- **Chapter 6** discusses some commonly encountered performance problems in a DB2 OLTP and BI environment, and describes scenarios for identifying and resolving such problems

- **Appendix A** describes the main performance enhancements in DB2 UDB ESE Version 8 and the latest performance enhancements in DB2 UDB Version 8.1.4.
- **Appendix B** describes the applications used in the problem scenarios. It includes scripts and sample code used in the various problem determination scenarios.

The team that wrote this redbook

This redbook was produced by a team of specialists from around the world working at the International Technical Support Organization, San Jose Center.

Nagraj Alur is a Project Leader with the IBM International Technical Support Organization, San Jose Center. He has more than 28 years of experience in DBMSs, and has been a programmer, systems analyst, project leader, consultant, and researcher. His areas of expertise include DBMSs, data warehousing, distributed systems management, and database performance, as well as client/server and Internet computing. He has written extensively on these subjects and has taught classes and presented at conferences all around the world. Before joining the ITSO in November 2001, he was on a 2-year assignment from the Software Group to the IBM Almaden Research Center, where he worked on Data Links solutions and an eSourcing prototype.

Peter Farrell is a Consulting IT Specialist in Australia. He has worked with computer systems for over 30 years and has extensive experience in operating systems and networking software development. He also has many years of experience running relational database benchmarks. He has over 15 years of experience in UNIX® technical support and has a particular interest in systems performance. He has worked at IBM for four years and before that was with Sequent® Computer Systems. His current areas of responsibility include benchmarking and performance tuning.

Philip Gunning is an independent DB2 consultant. He the founder of Gunning Technology Solutions, LLC. He has 17 years of experience in IT, and seven years with DB2 UDB. He holds a Masters degree from DeSales University. His areas of expertise include performance and tuning, database administration, and knowledge transfer. He has written extensively on DB2 UDB performance and is the author of the *DB2 UDB V8 Handbook for Linux, UNIX, and Windows*, IBM Press, June 2003. He is a member of the IDUG North America Conference Planning Committee and is a regular contributor to the db2-l list server and dbazine.com.

Saeid Mohseni is a DB2 education specialist with IBM Sweden. He has 23 years of experience in the Information Technology field, and 15 years of experience with the DB2 family of products. He holds a degree in Information Technology from Sweden. His areas of expertise include database administration/architect, performance tuning, database recovery, and system development using DB2.

Swaminaathan Rajagopalan is a Database Consultant with Wipro Technologies, a leading software services company in India. He has 7 years of IT experience, and hold a Masters Degree in Software Engineering from BITS Pilani, India. He is an IBM Certified solutions expert - DB2 UDB V7.1 Database Administration for OS/390®, and IBM certified solutions expert - DB2 UDB V7.1 Database Administration for UNIX, Linux, Windows and OS/2®. He has been working as a DB2 DBA on systems and applications for leading financial companies in the USA on assignment from Wipro Technologies. His areas of expertise include DB2 on OS/390, DB2 UDB on Linux, UNIX and Windows, DB2 for AS400, Oracle and Sybase. He has contributed articles to the Xephon DB2 update magazine.

In producing this redbook, we borrowed heavily from many IBM DB2 manuals, IBM classroom materials, IDUG presentations, Data Management Technical Conference presentations, IBM Press books, articles and other IBM Redbooks™, which are listed in “Related publications” on page 491. We hereby acknowledge their significant contributions.

Thanks in particular to the very constructive comments and contributions of Steve Rees, Adam Storm, Calisto Zuzarte, Berni Schiefer, Dwaine Snow, Marcia Miskimen, and Doreen Stolz-Hofmann.

Thanks also to the following people for their contributions to this project:

Adrian Chan
Leslie Cranston
Grant Hutchison
Srilata Kammila
Jason Racicot
Jeff Riihimaki
Kelly Rodger
Peter Shum
Roger Zheng
IBM Toronto Laboratory

Bruce Lindsay
Guy Lohman
IBM Almaden Research Center

Mark Latondre
IBM USA

Emma Jacobs
International Technical Support Organization, San Jose Center

Become a published author

Join us for a two- to six-week residency program! Help write an IBM Redbook dealing with specific products or solutions, while getting hands-on experience with leading-edge technologies. You'll team with IBM technical professionals, Business Partners and/or customers.

Your efforts will help increase product acceptance and customer satisfaction. As a bonus, you'll develop a network of contacts in IBM development labs, and increase your productivity and marketability.

Find out more about the residency program, browse the residency index, and apply online at:

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

We want our Redbooks to be as helpful as possible. Send us your comments about this or other Redbooks in one of the following ways:

- Use the online **Contact us** review redbook form found at:

  ibm.com/redbooks

- Send your comments in an Internet note to:

  redbook@us.ibm.com

- Mail your comments to:

  IBM Corporation, International Technical Support Organization
  Dept. QXXE Building 80-E2
  650 Harry Road
  San Jose, California 95120-6099

1

Introduction to performance management

In this chapter we provide a general overview of performance management concepts. We also describe the high-level tasks that DBAs typically perform to ensure that their DB2 environment is performing adequately and meeting previously agreed-to service level objectives.

The topics covered are:

- Performance management
- Types of monitoring
- Problem determination methodology

1.1 Introduction

One of the main objectives of an IT organization is to ensure that its infrastructure delivers the required performance to ensure that business objectives are continuously met in a constantly evolving and changing business environment.

This requires the IT professional to adopt a strategy that is both *proactive* and *reactive* to conditions and events that would tend to adversely impact IT systems.

The *proactive* effort involves a number of tasks, including the following:

- Capacity planning of IT resources
- Choosing the most effective IT architecture for the current and anticipated workload
- Adopting best practices in application design, development, and deployment
- Performing rigorous regression testing prior to deployment in a production environment
- Performing routine monitoring of key performance indicators to forestall potential performance problems, as well as gather information for capacity planning

The *reactive* effort involves having a well-defined methodology for identifying the root cause of a problem, and resolving the problem by applying best practices.

In the following sections, we introduce the concept of performance management, describe the different types of monitoring available, and discuss a typical methodology for effective performance problem determination.

1.2 Performance management

Most contemporary environments range from standalone systems to complex combinations of database servers and clients running on multiple platforms. Critical to all these environments is the achievement of adequate performance to meet business requirements. Performance is typically measured in terms of response time, throughput, and availability.

The performance of any system is dependent upon many factors including system hardware and software configuration, number of concurrent users, and the application workload.

Note: Performance management is a complex issue, and can be defined as modifying the system and application environment in order to satisfy previously defined *performance objectives*.

These performance objectives must be:

- **Realistic** in that they should be achievable given the current state of the technology available. For example, setting sub-second response times to process millions of rows of data is not achievable.
- **Reasonable** in that while the technology may be available, the business processes may not require stringent performance demands. For example, demanding sub-second response times for analytic reports that need to be studied and analyzed in detail before making a business decision could be considered unreasonable.
- **Quantifiable** in that the objectives must use quantitative metrics (numbers, ratios, percentages) instead of qualitative metrics (such as very good, average, and so on). An example of a quantitative metrics could be 95% of a particular transaction time must have sub-second response time, while a qualitative metric could be that system availability should be very high.
- **Measurable** in that you have to be able to measure the performance in order to determine conformance or non-conformance with performance objectives. Units of measurement include response time for a given workload, transactions per second, I/O operations, CPU use, or a combination of the above. Setting a performance objective of sub-second response times for a transaction is moot if there is no way to measure to determine whether this objective is being met.

Important: Without well-defined performance objectives, performance is a hit or miss exercise, with no way of delivering on any service level agreements that may be negotiated with users.

Figure 1-1 highlights the performance management cycle.

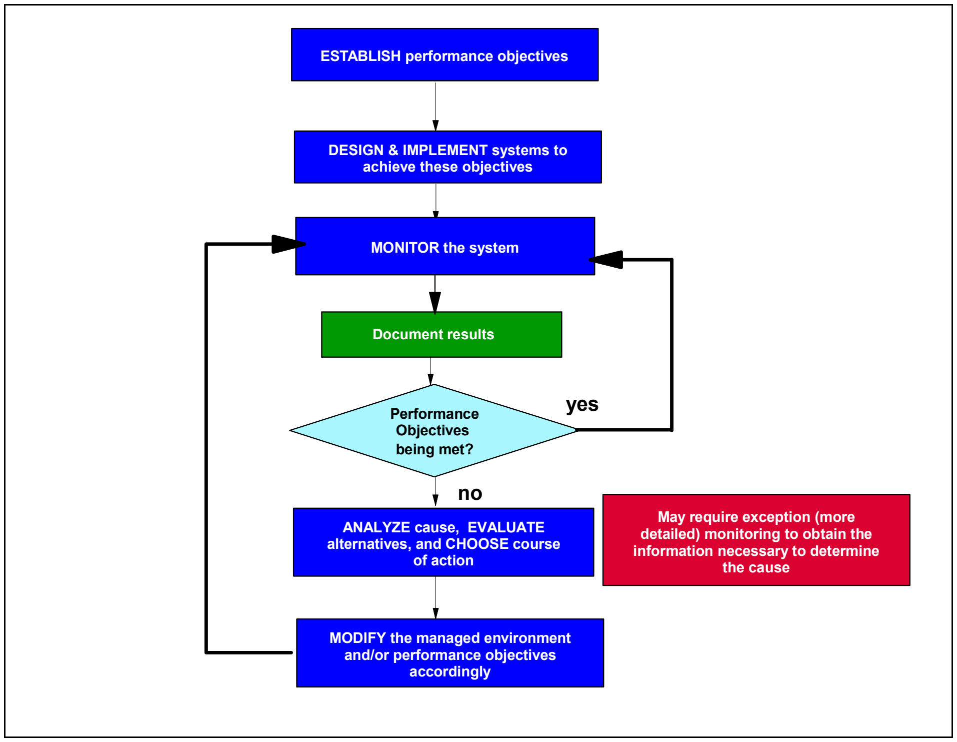

ESTABLISH performance objectives
DESIGN & IMPLEMENT systems to achieve these objectives
MONITOR the system
Document results
Performance Objectives being met?
yes
no
ANALYZE cause, EVALUATE alternatives, and CHOOSE course of action
May require exception (more detailed) monitoring to obtain the information necessary to determine the cause
MODIFY the managed environment and/or performance objectives accordingly

Figure 1-1 Performance management cycle

Performance management is an iterative process that involves constant monitoring to determine whether performance objectives are being met even as environments and workloads change over time. When performance objectives are not being met, then appropriate changes must be made to the hardware and/or software environment, as well as the performance objectives themselves, in order to ensure that they will be met.

From a database perspective, performance problems can arise out of a combination of poor application and system design, inadequate CPU, memory disk, and network resources, and suboptimal tuning of these resources.

Besides using monitoring to determine whether or not performance objectives are being met, monitoring is also used to:

- Assess the current workload of a system and track its changes over time for capacity planning purposes.

- Take a proactive approach to performance management by forestalling and resolving potential problems that could impede the achievement of performance objectives.
- React to unexpected problems by assisting in problem diagnosis.

Attention: In order to meet performance objectives currently and in the future, the DBA needs to develop and execute an appropriate monitoring strategy capable of delivering the required quality of service.

Important: Performance management can only be exercised in *controlled* environments such as production systems or regression systems.

Remember, “You can not manage what you can not control, and you can not control what you can not measure”.

Therefore, test and development systems by definition are inherently unmanageable.

1.3 Types of monitoring

There are typically three types of monitoring available to a DBA as follows:

- Routine monitoring
- Online/realtime event monitoring
- Exception monitoring

Each of these types of monitoring is briefly described in the following sections.

1.3.1 Routine monitoring

The objectives of routine monitoring are to:

- Collect information about the workload and stress on the system during periods of normal and peak periods for capacity planning purposes, as well as for identifying potential performance problems down the road
- Ascertain conformance of the system with performance objectives and record deviations, if any

The key to routine monitoring is that it involves analyzing the information collected over a period of time (long history), and then taking corrective action if required to address performance objectives. In other words, there can be a significant delay between information collection and a corrective response.

An example of the results of such monitoring could be a realization that the number of transactions has been growing steadily at a 1% rate every week, which would necessitate an upgrade of the server in 12 months in order to continue to meet response time objectives.

Another characteristic of such monitoring is the critical need to minimize the overhead it introduces, given its requirement to be running constantly or during peak periods.

In some literature, this type of monitoring is further subclassified into continuous monitoring (for normal loads) and periodic monitoring (for peak loads).

From a DB2 perspective, routine monitoring can help identify the root causes of such potential performance problems as:

- Buffer pool size
- Dynamic cache size
- Heap sizes
- Locklist and maxlocks sizes
- Lock mode and isolation issues
- Disorganized table spaces
- Outdated runstats
- Long running SQL
- Log and table space utilization

1.3.2 Online/realtime event monitoring

The objective of online/realtime event monitoring is to be on the lookout for *specific* events that may either identify a specific problem, or portend problems in the near to immediate future, in order to take prompt corrective action (the near to immediate future implies minutes rather than hours).

The key to this type of monitoring is that it involves looking for specific events in a short interval of time (short history) that are known to degrade performance, and having the option to take prompt corrective action to rectify the problem. In other words, there probably needs to be a very short delay between information collection and a corrective response. One example of such an event is the occurrence of an excessive number of deadlocks in a short period of time, a problem that would need to be addressed promptly in order to ensure that business objectives are not being compromised.

Here too, the need to minimize the overhead of such monitoring is critical, given that most problems manifest themselves at peak loads.

From a DB2 perspective, online/realtime event monitoring can help identify potential performance problems such as:

- Deadlocks
- Long waits and timeouts
- Long running SQL

1.3.3 Exception monitoring

Exception monitoring is required when you discover or suspect a problem, and need to identify its root cause in order to apply the appropriate corrective action to fix the problem.

Unlike routine and event monitoring, which are planned occurrences and are designed to have low overheads on the managed system, exception monitoring is driven by problem situations and may impose significant overheads on the managed system. An example of a need for exception monitoring is when the administrator receives a significant number of user complaints about degraded response times, or inability to access the application. The administrator then needs to initiate a series of monitoring actions to home in on the root cause of the problem. This typically involves coming up with a set of hypotheses that could account for the perceived behavior, and then systematically verifying each one in turn until the problem is diagnosed.

From a DB2 perspective, exception monitoring can apply to any of the items identified via routine and online/realtime event monitoring.

1.4 Problem determination methodology

Users may experience performance problems for reasons ranging from:

- Network connectivity and bandwidth constraints
- System CPU, I/O and memory constraints
- Software configuration limitations and constraints
- Poor systems administration skills
- Poor application design
- Poor assumptions about the workload

Given the multitude of possible causes of poor performance, a systematic and consistent approach to problem diagnosis is recommended to ensure prompt and effective resolution of performance problems.

Figure 1-2 on page 8 describes a typical sequence of steps to be followed when diagnosing performance problems.

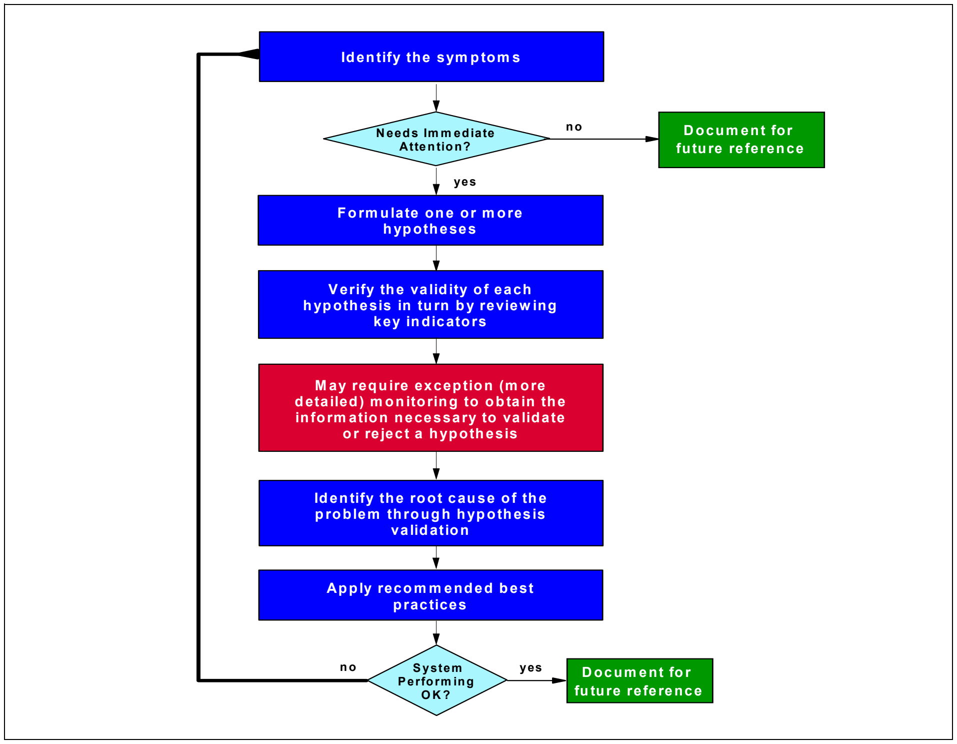

Identify the symptoms
Needs Immediate Attention?
no
Document for future reference
yes
Formulate one or more hypotheses
Verify the validity of each hypothesis in turn by reviewing key indicators
May require exception (more detailed) monitoring to obtain the information necessary to validate or reject a hypothesis
Identify the root cause of the problem through hypothesis validation
Apply recommended best practices
no
System Performing OK?
yes
Document for future reference

Figure 1-2 A typical problem determination methodology

The entire sequence of steps is triggered by events such as a user complaining about poor response times, error messages appearing on users' screens, alerts or notifications on the DBA console, and alerts in routine monitoring reports.

These symptoms must be evaluated for criticality, as shown by the decision box ("Needs immediate attention") in Figure 1-2.

- Symptoms that are sporadic and non-disruptive need no immediate action, other than to potentially trigger exception, online/realtime, or additional routine monitoring to gather additional information for possible corrective action in the future.
- Symptoms that recur frequently and disrupt business processes require prompt attention to avoid adverse business impact. We cover some of these scenarios in this book.
- Catastrophic events such as a failure of the system, the application server, or the database server also need immediate attention such as an immediate restart. These scenarios are *not* discussed in this book.

Based on the symptoms and a knowledge base of prior experiences (both external and internal), you should formulate one or more hypotheses as the potential root cause of the problem.

Each hypothesis should then be tested in turn using all available metrics associated with the application under consideration — this includes system resources, network resources, Web application server resources, and database server resources. Sometimes, the metrics available from routine monitoring and online/realtime event monitoring may be inadequate to validate or reject a particular hypothesis. In such cases, you may have to request additional diagnostic information through more detailed monitoring levels either on the production system itself, or an available comparable regression system. Such monitoring is often referred to as *exception monitoring*.

Once a hypothesis is validated and the root cause problem has been identified, best practices specific to the root cause problem can be applied to attempt to resolve the problem.

Important: Best practices guidelines are based on user experiences for a given workload and environment, and may or may not provide beneficial results in your particular environment. Therefore, a thorough understanding of the fundamentals of the technical architecture and design is required to explore other alternatives, when the documented best practices fail to provide relief.

Problem resolution in such cases tends to be an iterative process, where the application of a best practice may result in the manifestation of new symptoms, and the formulation of a fresh set of hypotheses.

Once the root cause problem has been resolved, then the steps executed and the knowledge gained should become part of the knowledge base to assist in resolving future problem situations.

Attention: When applying best practices recommendations, it is vital that changes be implemented *one* at a time and the impact measured before embarking on further changes. Implementing multiple changes simultaneously may make it difficult to assess the impact of a specific change and develop a useful knowledge base for future performance tuning efforts.

A knowledge base of all successful and unsuccessful changes in your environment should be accumulated to develop best practices and recommendations for your own, unique environment.

Figure 1-3 describes one possible DB2 application environment, having both remote and local clients accessing a DB2 database. While this environment shows the Web application server and the database server on different systems, other application environments may be simpler or more complex depending upon an organization's unique configuration and application workload requirements.

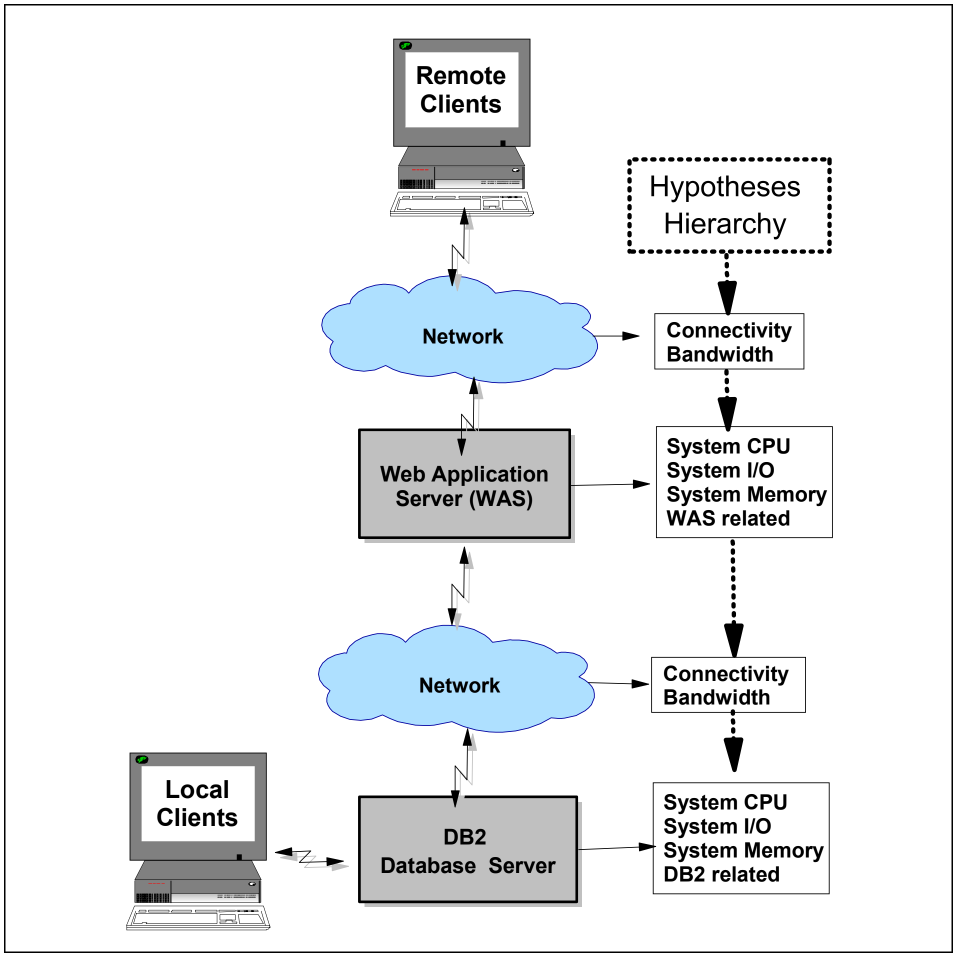


Figure 1-3 A typical DB2 application environment and hypotheses hierarchy

Figure 1-3 also describes the hypotheses validation hierarchy that should typically be followed during problem diagnosis of DB2 applications in general.

Note: Depending upon the triggering event of the performance problem, it may be possible to skip certain hypotheses validation altogether—for example, when an explicit alert about a lock escalation threshold being tripped is the triggering event, you can ignore hypotheses such as network connectivity and bandwidth constraints, Web application server constraints, CPU and I/O constraints (in both the Web application server and the DB2 database server) as a potential cause of the problem.

Assuming that a performance problem such as erratic or poor response times has been reported by a remote user/client, the problem determination methodology should systematically home in on the potential cause of the problem via the hierarchy shown in Figure 1-3 on page 10.

For the environment shown in Figure 1-3, the process should sequentially involve eliminating the cause of the problem as being network (between the client and the Web application server)-related, Web application server system (CPU, I/O, memory)-related, Web application server-related (configuration settings), network-related, database server system (CPU, I/O, memory)-related, or database server (configuration settings, routine DBA maintenance activities such as collecting statistics or reorganizing tables, and finally application design)-related.

Important: The Figure 1-3 sequence is strongly recommended to ensure that needless effort is not spent by the DBA on troubleshooting DB2, when the root cause of the performance problem experienced by the user potentially exists elsewhere.

For example, network bandwidth constraints or resource contention in the Web application server can manifest as erratic or poor response times for a user of a DB2 application, even when the DB2 system and application is tuned optimally.

2

DB2 UDB architecture overview

In this chapter we provide a general overview of the architecture of DB2 UDB from a performance management perspective. We also describe the flow of a transaction/query as it interacts with various processes and resources in DB2, and identify potential performance bottlenecks along the way—both in a single user environment with no contention, and in a multi-user environment involving contention for various resources. Also described are the key parameters and monitoring facilities available in DB2 to manage application and system performance.

The topics covered are:

- Main components of DB2
- Single-user transaction/query flow
- Multi-user (concurrent) transaction/query flow
- Key performance knobs
- Performance monitoring facilities in DB2

2.1 Introduction

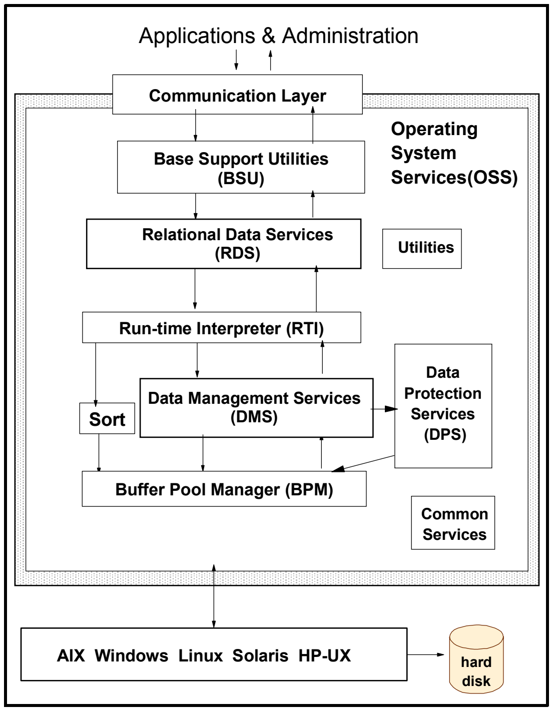
Critical to performance tuning of a DB2 environment is an understanding of the main components of DB2, their functionality, and how they interact with each other. Also important is to understand the flow of an transaction within DB2 and appreciate the various performance drivers within each component, as well as key tuning knobs available to impact their performance.

2.2 Main components of DB2

We describe the main components of DB2 from multiple perspectives including a high level overview of the main DB2 components, a high level overview of DB2 architecture and processes, a detailed view of the DB2 process model, and an overview of the memory model.

2.2.1 High level overview of main DB2 components

Figure 2-1 provides a high level overview of the main DB2 components.



Applications & Administration
Communication Layer
Operating System Services(OSS)
Base Support Utilities (BSU)
Relational Data Services (RDS)
Utilities
Run-time Interpreter (RTI)
Data Protection Services (DPS)
Sort
Data Management Services (DMS)
Buffer Pool Manager (BPM)
Common Services
AIX Windows Linux Solaris HP-UX
hard disk

Figure 2-1 High level overview of main DB2 components

A brief overview of each of these components, the name of the engine[1] (<...>), and their interaction with each other follows:

- **Applications** may communicate with the DB2 server over local or remote connections using a wide variety of protocols and interfaces. Each client application is linked with the DB2 client library and communicates with the DB2 server using shared memory and semaphores for local clients, and communication protocols such as TCP/IP and APPC for remote clients.
- **Base Support Utilities (BSU)** <`sqe`> supports tasks associated with starting DB2 (db2start), creating a database, connecting to DB2 (connect), engine dispatchable unit (EDU) services, and routing.
- **Relational Data Services (RDS)** <`sqr`> supports many operations for processing data such as joins, grouping, aggregation and math. RDS includes the DB2 optimizer which performs access path optimization and generates compiled SQL sections. The RDS also includes catalog services which manages the metadata describing the tables, indexes and views, as well as statistics on the data.

  Note: SQL predicates evaluated by RDS are sometimes referred to as *non-sargable* or *residual* predicates.

- **Run-time Interpreter (RTI)** <`sqri`> executes the compiled sections generated by RDS using one or more RDS operations such as joins, aggregation and grouping.
- **Data Management Services (DMS)** < `sqd`> provides the data structures for tables, long fields, large objects, and indexes. DMS also performs insert, update and delete operations on these structures. Some of the components included in DMS are the Table Manager <`sqd`>, Index Manager <`sqx`>, Long object Manager <`sqdl`>, and Large object Manager <`sqlx`>.

  Note: SQL predicates evaluated by DMS are called *data sargable* or *index sargable* predicates.

- **Sort** is performed by Sort List Services (SLS) < `sqs`> .
- **Data Protection Services (DPS)** <`sqp`> supports transaction management and logs every operation that modifies the database for crash recovery. DPS includes locking and logging services.
- **Buffer Pool Manager (BPM)** <`sqb`> provides look-aside caching which improves performance by caching portions of the database in memory to

[1] The executable module.

avoid I/O, retrieving data before it is requested, and writing out modified data asynchronously.

- **Operating System Services (OSS)** <`sqo`> provides a common interface for functions such as file reading and writing, shared memory, and latching for the rest of the database engine.
- **Utilities** support tasks such as load <`squ`>, backup <`sql`>, restore <`sql`>, rollforward <`sql`>, import <`sqs`> and export <`sqs`>.
- **Common services** provides functions such as system monitor <`sqm`> for monitoring and tuning, and configuration services <`sqf`>.

When applications connect to the database using local or remote connections, an EDU (`db2agent`) is created by BSU to perform the requested work. The application's SQL statement is optimized by RDS, which generates compiled SQL sections detailing the work to be done. The RTI executes these compiled sections accessing appropriate tables/indexes via DMS.

DMS accesses data from the buffer pool and/or tables/indexes and returns the results to the RTI after applying any sargable predicates. RTI applies any non-sargable predicates and performs any required RDS operations, including applying sorts if required, before returning the final result back to the application via the RDS.

Attention: This is an oversimplification of the components available in DB2 and the interaction between them. The component functionality described here is also incomplete and oversimplified and is only meant to give readers a feel for some of the components and their interrelationships.

From a performance perspective, one particular point may be worth noting: both the RDS and DMS evaluate SQL predicates which filter the data returned to the application. However, there is a performance advantage to predicates being evaluated by the DMS rather than the RDS, since non-qualifying rows can be rejected earlier.

With a non-sargable predicate (evaluated by the RDS), every row retrieved by DMS has to be passed to the RDS for possible qualification, which adds to the path length of the query. By extension, the worst case scenario is when the application performs the predicate evaluation in user code rather than as a predicate in the SQL statement.

Sargable and non-sargable predicates are discussed in greater detail in "Limit the volume of data returned - columns and rows" on page 160.

2.2.2 High level overview of DB2 architecture and processes

Figure 2-2 provides a high level overview of DB2 architecture and processes.

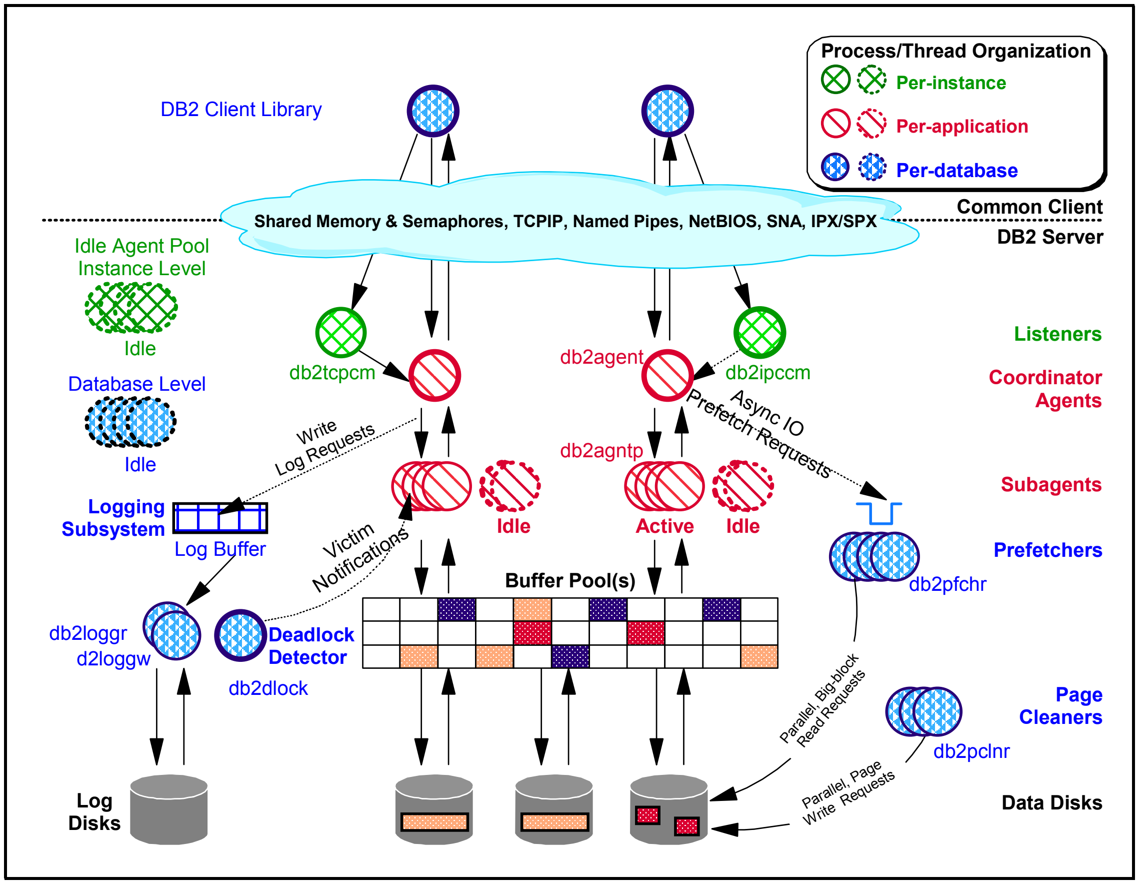


Figure 2-2 High level overview of DB2 architecture and processes

As mentioned earlier, each client application is linked with the DB2 client library, and communicates with the DB2 Server using shared memory for local clients, or a communication protocol such as TCP/IP for remote clients. Each circle in Figure 2-2 is an EDU, which are implemented as threads on Windows (all under a single process db2sysc) and processes on Linux and UNIX.

The key elements in Figure 2-2 are briefly described here; refer to 3.4, “System environment considerations” on page 198 for a more detailed explanation.

DB2 agents

DB2 agents include coordinator agents and subagents, and are the most common type of DB2 processes that carry out the bulk of SQL processing on behalf of applications. DB2 assigns a coordinator agent with an application, and this agent coordinates the communication and processing for this application.

Note: If intra-partition parallelism is disabled (this is the default), then the coordinator agent performs all the application's requests. If intra-partition parallelism is enabled, then DB2 assigns a set of subagents to the application to work on processing the application requests.

The following database manager and database configuration parameters determine how many database agents are created and how they are managed.

- **`MAXAGENTS`** database manager parameter is the maximum number of database manager agents that can be working at any one time, including coordinator agents, subagents, inactive agents, and idle agents.
- **`MAXCAGENTS`** database manager parameter is the maximum number of database manager agents that can be concurrently executing a database manager transaction.
- **`MAXAPPLS`** database configuration parameter is the maximum number of applications that may 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. If set to `AUTOMATIC`, you can create any number of databases, and memory usage will grow accordingly.

  Note: The default setting for **`MAXAPPLS`** in DB2 Version 8 is **`AUTOMATIC`**.

- **`MAX_CONNECTIONS`** database manager parameter is the maximum number of client connections to a database. When the connection concentrator is enabled, this parameter works a little differently as discussed in "Connection concentrator" on page 19.
- **`MAX_COORDAGENTS`** database manager parameter is the maximum number of database manager coordinator agents or application requests that can be processed at any one time. One coordinating agent is acquired for each local or remote application that connects to a database or attaches to an instance.
- **`NUM_POOLAGENTS`** database manager parameter is 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.`**

- `NUM_INITAGENTS` database manager parameter is the total number of worker agents that are created when the database manager is started. This speeds up performance for initial queries. The worker agents all begin as idle agents.

The notion of pooling agents at the database level was added in DB2 UDB Version 8.1. In reality, the database level idle pool is comprised of one or more pools for particular groups of applications. Initially, when a system starts up, there is only one Application Group. As more connections are added, DB2 creates additional Application Groups and creates an idle pool for each new Application Group—this is done automatically and transparently by DB2.

There are typically 100-200 applications per Application Group.

Note: Application Groups help spread out access to critical resources such as shared SQL work space across agents. This can help reduce contention in very high user population environments.

Connection concentrator

For internet applications with many simultaneous user connections, the connection concentrator may improve performance by allowing many more client connections to be processed efficiently. It also reduces memory use for each connection, and decreases the number of operating system-level context switches. This feature is intended for short-running Web applications, and is not intended for long-running applications such as the production of big reports.

Note: The connection concentrator feature is new in DB2 Version 8 and is off by default.

Figure 2-3 on page 20 describes the connection concentrator concept.

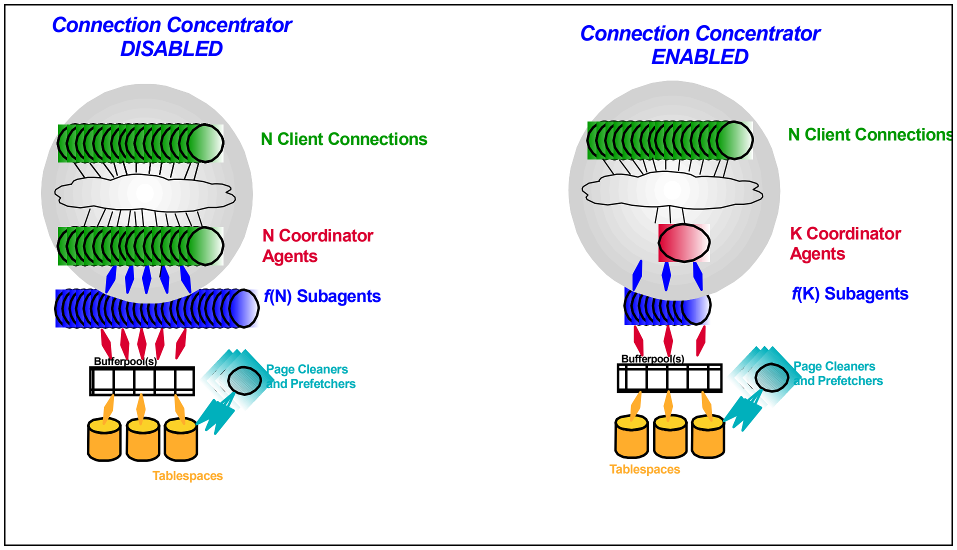


Figure 2-3 Connection concentrator concept

Connection concentrator is enabled when **`MAX_CONNECTIONS`** is greater than **`MAX_COORDAGENTS.`** In this case, there may be more connections than coordinator agents to service them.

With connection concentrator disabled, each connected application always has an agent assigned to it—there is a one-to-one relationship between connections and agents, as shown in the left half of Figure 2-3 on page 20.

With connection concentrator enabled, there can be a many-to-one relation between connections and DB2 agents, as shown in the right half of Figure 2-3 on page 20.

Figure 2-4 on page 21 describes the operation of connection concentrator.

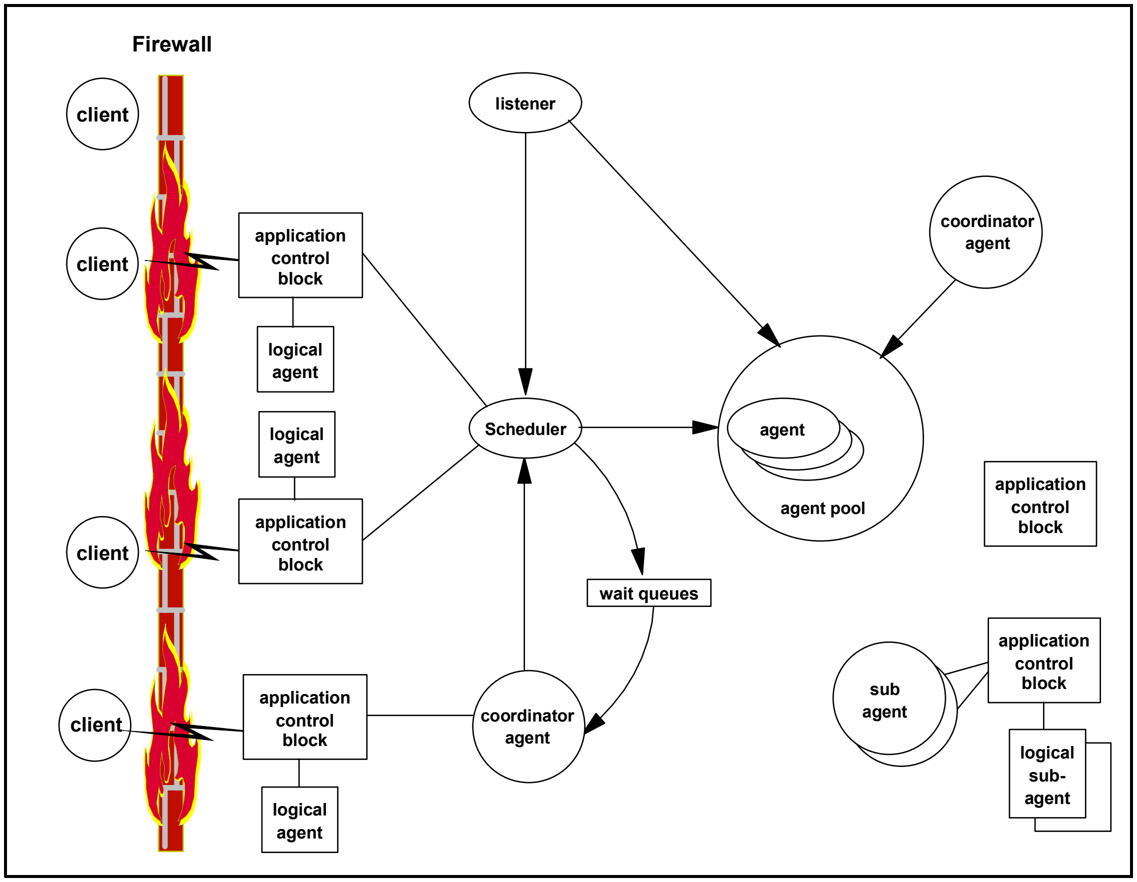


Figure 2-4 Connection concentrator operation

When a client application connects to a database, an application control block is associated with it.

When the client begins, a transaction the following occurs:

1. One or more dispatchers try to find/create an agent to work on it. If there is no free idle agent, or if an agent cannot be created because the configuration limit on the number of coordinator agents has been reached, the request is queued.
2. When a transaction ends for another application, the coordinating agent associated with it becomes available and serves the next transaction on the queue, regardless of connection. Coordinating agents first look to their own application group for new transaction requests.

Note: There is one queue per Application Group, and each agent is associated with a particular application group.

Also, to ensure that no single application group dominates system resources, there is a mechanism that allows agents to migrate from across application groups over time.

A coordinator agent will return to the pool if there is no application waiting to be served at the time the coordinating agent finishes the transaction. A coordinating agent will terminate only after it has been idle and waiting in the pool for an amount of time determined by the logical agent scheduler. A subagent will return to the agent pool after servicing a request, so that it can be used for other requests.

For more details on the connection concentrator, refer to *DB2 Connect User's Guide*, SC09-4835, and *DB2 UDB Administration Guide: Performance* SC09-4821.

Buffer pools

A *buffer pool* is an area of memory into which database pages of user table data, index data, and catalog table data are temporarily moved from disk storage. DB2 agents read and modify data pages in the buffer pool. The buffer pool is a key influencer of overall database performance, because data can be accessed much faster from memory than from a disk. If more of the data needed by applications is present in the buffer pool, then less time would be needed to access this data, thereby improving performance.

Buffer pools can be defined with varying page sizes including 4 K, 8 K, 16 K and 32 K.

Block-based buffer pools

In DB2 UDB Version 8, prefetching on certain platforms can be improved by creating block-based buffer pools.

By default, 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.

Prefetching code recognizes when a block-based buffer pool is available and will use block I/Os to read multiple pages into the buffer pool in a single I/O, thereby significantly improving the performance of prefetching.

A block-based buffer pool consists of both a page area and a block area.

- The page area is required for non-sequential prefetching workloads.
- The block area consists of blocks where each block contains a specified number of contiguous pages—referred to as the block size.

Prefetchers

Prefetchers are present to retrieve data from disk and move it into the buffer pool before applications or utilities 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 prefetchers.

With prefetch, DB2 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.

`NUM_IOSERVERS` is the database configuration parameter that controls the number of prefetchers.

Page cleaners

Page cleaners are present to make room in the buffer pool, before agents and prefetchers read pages from disk storage and move them into the buffer pool. For example, if an application has updated a large amount of data in a table, many of the updated data pages in the buffer pool may not yet have been written to disk storage—such pages are called “dirty” pages. Since prefetchers cannot place fetched data pages on the dirty pages in the buffer pool, these dirty pages must first be flushed to disk storage and become “clean” pages, so that prefetchers can find room to place fetched data pages from disk storage.

Note: When a page cleaner flushes a dirty page to disk storage, the page cleaner removes the dirty flag but leaves the page in the buffer pool. This page will remain in the buffer pool until a prefetcher or a DB2 agent steals it.

Page cleaners are independent of the application agents that read and write to pages in the buffer pool.

`NUM_IOCLEANERS` is the database configuration parameter that controls the number of page cleaners.

Logs

Changes to data pages are logged. Agent processes updating a data record in the database update the associated page in the buffer pool, and write a log record into a log buffer.

To optimize performance, like the updated pages in the buffer pool which are asynchronously written to the disk by the page cleaners, the log records in the log buffer are also written asynchronously to disk by the logger. The logger and the buffer pool manager cooperate and ensure that an updated page is not written to disk storage before its associated log record is written to the log. This ensures database recovery to a consistent state from the log in the event of a crash such as a power failure.

Log buffers are flushed to disk under the following conditions:

- Before the corresponding data pages are written to disk—this is called write-ahead logging.
- On a COMMIT, when the **`MINCOMMIT`** database configuration parameter is set to 1 (default).
- When **`MINCOMMIT`** is set to greater than 1, commit grouping is in effect and application commit requests are held (wait) until either one second has elapsed or the number of commit requests equals the value of this parameter.
- When the log buffer is full.

However, the logger also attempts to keep buffers available for new log records to minimize log waits for buffers (which can have a significant, detrimental impact on performance). It does this by checking for log buffers ready to be written and writing them out as soon as the logger has just completed a previous write of log buffers.

Deadlock detector

A deadlock occurs when one or more applications require access to a resource that is currently locked by the other application(s).

In order to avoid extended waits in the event of a deadlock, DB2 uses a background process called the deadlock detector (db2dlock) to identify and resolve these deadlocks. The deadlock detector becomes active periodically as determined by the **`DLCHKTIME`** database configuration parameter. When the deadlock detector encounters a deadlock situation, one of the deadlocked applications will receive an error code, and the current unit of work for that application will be rolled back automatically by DB2. When the rollback is complete, the locks held by this chosen application are released, thereby allowing other applications to continue.

Selecting the proper interval for the deadlock detector ensures good performance. Too short an interval causes unnecessary overhead, while too long an interval allows a deadlock to delay processes unnecessarily.

Note: DB2 UDB Version 8 enhancements to the deadlock Event Monitor now provide more information to help the DBA determine the cause of the deadlocks. For example, the deadlock Event Monitor now identifies the specific statements involved in a deadlock, and pinpoints the specific locks held by each application involved in the deadlock.

Disks

Data placement, and the number and type of disks, can play a significant role in the performance and availability of a database application.

The overall objective is to spread the database effectively across as many disks as possible to try to minimize I/O wait. Operating systems and file systems support a variety of data placement strategies including mirroring, striping, and RAID devices, and DBAs need to understand the advantages and disadvantages of disk capabilities in order to design the most appropriate strategy for their environments.

2.2.3 Process model

The EDUs described in Figure 2-2 on page 17 are implemented as processes in UNIX, and threads in a single process (db2sysc) in Windows.

Besides the processes described in Figure 2-2 on page 17, there are a number of other EDUs in DB2 as well.

It is helpful to know the names and functions of these processes, because they may help in problem diagnosis. For example, high CPU utilization for a particular DB2 process may pinpoint the need for a reconfiguration of a particular database manager or database configuration parameter.

Important: Do not directly terminate DB2 processes in a healthy DB2 environment, since it will cause undesirable effects in DB2. Use the **`db2stop`** command to terminate DB2.

The objective of this section is to provide an understanding of the EDUs involved, but not manipulate them directly.

Refer to *DB2 UDB Version 8 Administration Guide: Performance*, SC09-4821, for more detailed information about DB2 processes.

Figure 2-5 on page 26 shows a simplified view of the DB2 process model.

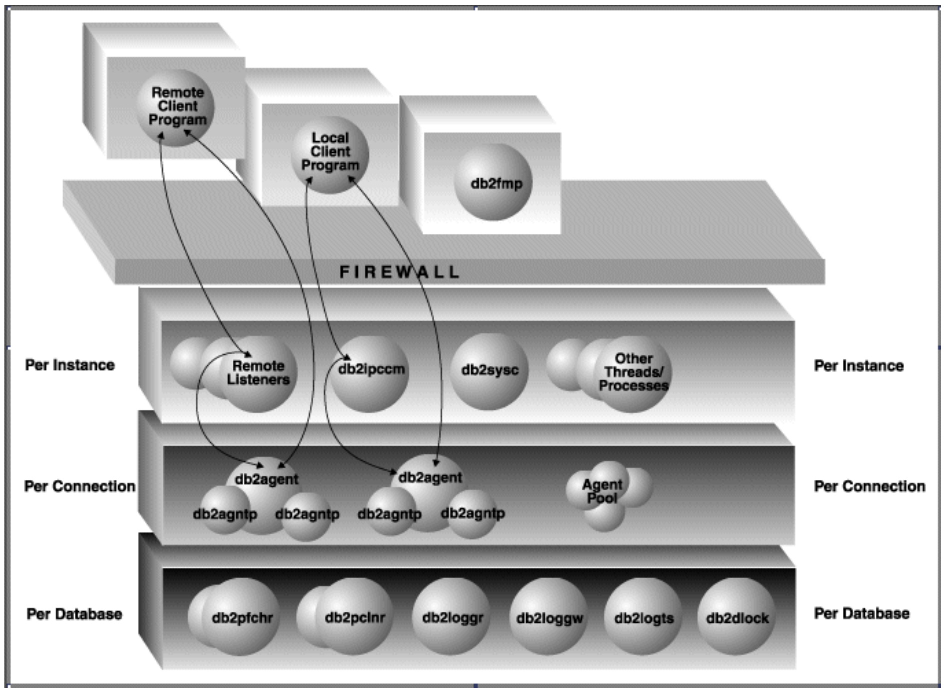


Figure 2-5 Simplified view of the DB2 UDB Process Model

The processes shown in Figure 2-5 are briefly described, followed by Table 2-1 on page 28, Table 2-2 on page 28, and Table 2-3 on page 29 describing other DB2 processes of interest.

Client program

A client program may run remotely or on the same machine as the database server. Client programs establish contact through a listener process (db2icpcm, or db2tcpcm, depending upon the communication protocol), and are assigned to a coordinator agent (db2agent) on a successful connection. They db2fmp process shown is a user-created, user-defined function or a stored procedure.

Firewall

DB2 architecture provides a firewall so that applications run in a different address space from DB2. The firewall isolates and protects the database and the database manager from applications, stored procedures (non-fenced), and user defined functions (non-fenced) that may crash the database manager.

Listeners

A client program makes initial contact with communication listeners processes which are started when DB2 is started—these processes are associated with the DB2 instance. There is a listener for each configured communication protocol, and an interprocess communication (IPC) listener for local client programs as follows:

- db2icpcm for local client connections
- db2tcpcm for TCP/IP connections
- db2tcpdm for TCP/IP discovery tool requests

Note: The db2sysc process is the main DB2 system controller or engine; without this process, the database server cannot function.

db2agent and db2agntp

An *agent* is the worker that performs all database operations on behalf of an application. All connection requests from client applications, whether they are local or remote, are allocated a corresponding coordinator agent (db2agent). When the coordinator agent is created, it performs all database requests on behalf of the application.

When the **`INTRA_PARALLEL`** database manager configuration parameter is enabled, the coordinator agent distributes the database requests to subagents (db2agntp). The coordinator agent handles all database requests behalf of its application by coordinating subagents (db2agntp) that perform the actual requests on the database.

Agent pool

Agent creation and destruction is an expensive operation, and DB2 provides an agent reuse mechanism by implementing an agent pool to which agents or subagents are released when an application completes. These agents or subagents are then available to be reused for requests for coordinator agents on behalf of a client program, or for subagents on behalf of existing coordinator agents.

Table 2-1, Table 2-2 on page 28, and Table 2-3 on page 29 describes *some* of the more important processes visible during different states of the DB2 environment.

- Table 2-1 describes processes per instance when there are no connections or active databases.
- Table 2-2 on page 28 describes *additional* processes per instance when there are connections present.
- Table 2-3 on page 29 describes processes for each active database, in addition to the instance and connection processes.

For a more complete list of DB2 processes and an explanation of their importance, refer to the article “Everything You Wanted to Know About DB2 Universal Database™ Processes” by Dwaine Snow and Raul F. Chong, which is available at:

http://www7b.boulder.ibm.com/dmdd/library/techarticle/0304chong/0304chong.html

Table 2-1 DB2 processes per instance - no connections & no active databases

Process name	Brief description
db2cart	Determines when a log file can be archived and invokes the user exit to do the actual archiving. There is one db2cart process per instance, but it only runs if there is at least one database in the instance which has USEREXIT enabled.
db2fmtlg	Pre-allocates log files in the log path when the database is configured with LOGRETAIN ON and USEREXIT OFF.
db2gds	The DB2 Global Daemon Spawner process that starts all DB2 EDUs (processes) on UNIX. There is one db2gds per instance or database partition.
db2disp	The DB2 agent dispatcher process, which dispatches client connections between the application assigned to the client connection and the available coordinating agents. This process only exists when connection concentrator is enabled.
db2sysc	The main DB2 system controller or engine. Without this process, the database server cannot function
db2wdog	The DB2 watchdog handles abnormal terminations; this only applies to UNIX.
db2govd	The DB2 Governor, which is a reactive governing process. If the DB2 governor is enabled, this process takes snapshots at the interval specified in the governor configuration file and checks the snapshots against all configured rules. If a rule is broken, it takes the specified action.

Table 2-2 Additional DB2 processes per instance with connections

Process name	Brief description
db2agent	DB2 coordinator agent, which performs all database requests on behalf of an application.
db2agnsc	The parallel recovery agent, which is used during roll forward and restart recovery to perform the actions from the logs in parallel.

Process name	Brief description
db2agnta	An idle subagent that was used in the past by a coordinator agent and is still associated to that coordinating agent process.
db2agntp	A subagent that is currently performing work on behalf of the coordination agent it is associated with. This subagent is enabled when the INTRA_PARALLEL database manager configuration parameter is set to YES.
db2ipccm	IPC communication manager is the interprocess communication listener for local client connections.
db2tcpcm	TCP communication manager works as a communication listener for TCP/IP connection requests.
db2tcpdm	Communication listener for TCP/IP discovery requests. Discovery requests are made by the configuration assistant when it searches the network for remote DB2 servers and their databases.
db2snacm	SNA/ APPC communication manager, which works as a communication listener for SNA/ APPC connection requests.

Table 2-3 Additional DB2 processes for each active database

Process name	Brief description
db2dlock	Local deadlock detector; there is one per database partition. It scans the lock list and looks for deadlock conditions.
db2estor	Used to copy pages between the database buffer pools and extended storage.
db2event	The Event Monitor process. There is one db2event process per active Event Monitor, per active database. These processes capture the defined “events” and write them to the output file specified for the Event Monitor.
db2loggr	The database log reader, which reads the database log files during transaction processing, restart recovery, and roll forward operations.
db2loggw	The database log writer, which flushes the log records from the log buffer to the log files on disk.
db2pclnr	The buffer pool page cleaners, which asynchronously write dirty pages from the buffer pool(s) back to disk.
db2pfchr	The buffer pool prefetchers, which read data and index pages from disk into the database buffer pool(s) before it is read on behalf of applications.

Process name	Brief description
db2logts	This process is used for collecting historical information about which logs are active when a table space is modified. This information is recorded in the DB2TSCHG.HIS file in the database directory. It is used to speed up table space roll forward recovery.

2.2.4 Memory model

Different applications use memory in different ways. For example, some applications may use the operating system cache for file handling, while the database manager uses it own buffer pool for data caching. Therefore, in order to achieve good performance, it is important to balance overall memory usage on the system to minimize paging, as described in 5.2.2, "Memory considerations" on page 335 and 5.3.2, "Memory considerations" on page 383. This requires a good understanding of how DB2 organizes memory.

Memory allocation is based on database manager and database configuration parameters. These parameters may specify hard or soft upper limits, lower bound values, or let DB2 automatically allocate memory resources as required via the `AUTOMATIC` setting. Depending upon the area of memory involved, DB2 allocates and deallocates memory at different times; some of the values of these parameters can be changed online to take immediate effect.

Figure 2-6 on page 31 shows the different portions of memory that the database manager allocates for various uses.

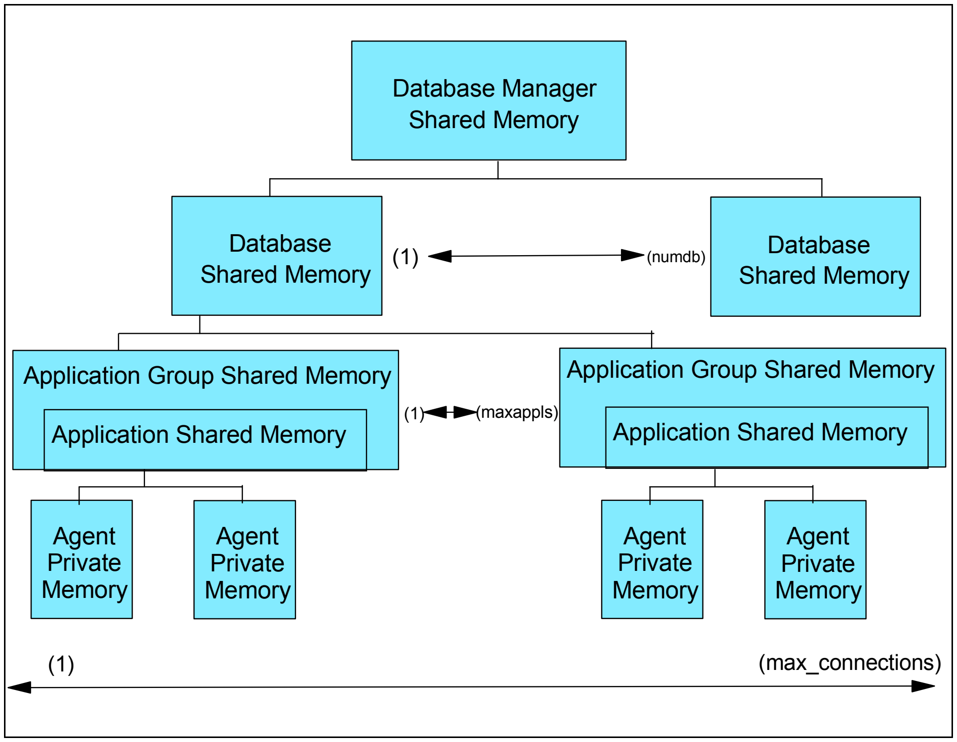


Figure 2-6 Types of memory used by DB2

There are a few special types of shared memory *not* shown in Figure 2-6, as follows:

- Agent/Local Application Shared Memory (database manager configuration parameter **`aslheapsz`**), which is attached by coordinating agents servicing local applications. It is used for SQL request/response communications.
- UDF/Agent Shared Memory, which is attached by agents running a fenced UDF or stored procedure. It is used as a communications area.
- Extended buffer pool, where a typically huge region (far in excess of 4 GB) of shared memory is used as an extended buffer pool. Agents, prefetchers, and page cleaners are not permanently attached to it. They attach to individual segments within it as needed.

Figure 2-7 on page 32 provides a more detailed view of the main elements that comprise each type of memory, and the database manager or database configuration parameters associated with each type of memory.

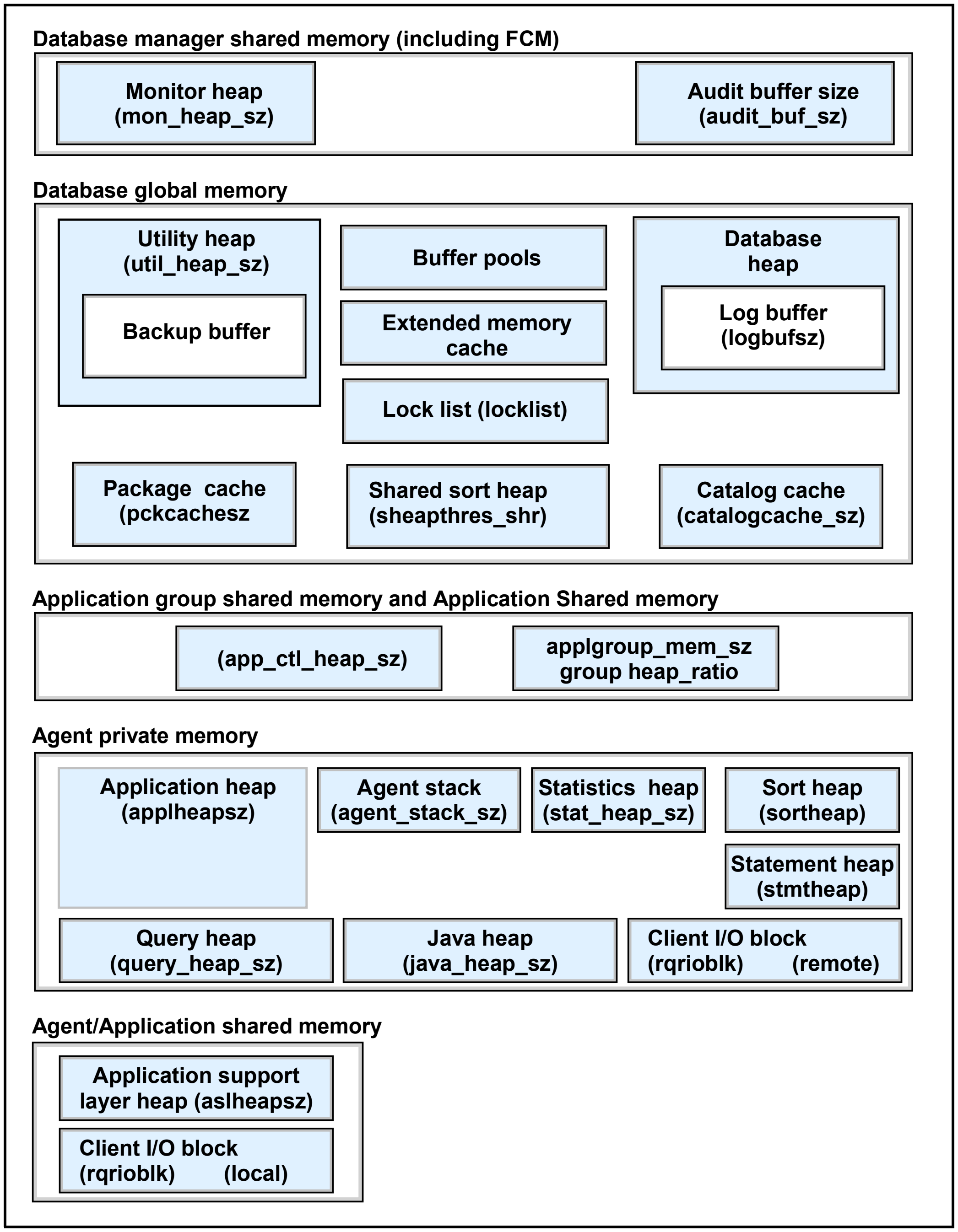

Database manager shared memory (including FCM)
Monitor heap (mon_heap_sz)
Audit buffer size (audit_buf_sz)
Database global memory
Utility heap (util_heap_sz)
Backup buffer
Buffer pools
Extended memory cache
Lock list (locklist)
Database heap
Log buffer (logbufsz)
Package cache (pckcachesz
Shared sort heap (sheapthres_shr)
Catalog cache (catalogcache_sz)
Application group shared memory and Application Shared memory
(app_ctl_heap_sz)
applgroup_mem_sz group heap_ratio
Agent private memory
Application heap (applheapsz)
Agent stack (agent_stack_sz)
Statistics heap (stat_heap_sz)
Sort heap (sortheap)
Statement heap (stmtheap)
Query heap (query_heap_sz)
Java heap (java_heap_sz)
Client I/O block (rqrioblk) (remote)
Agent/Application shared memory
Application support layer heap (aslheapsz)
Client I/O block (rqrioblk) (local)

Figure 2-7 The memory used by the database manager

In the following subsections, we describe each type of memory in more detail and discuss their allocation duration, allocation sizes and limits, overflows into and out of each heap, and the ability to modify them online for immediate effect.

Important: Each type of memory comprises heaps which contribute to the total memory allocated at the appropriate time; shared memory is allocated in *blocks*, while private memory uses *heaps*. For example, database shared memory is allocated *in full* at first connection to the database by an application or when the database is activated. If database shared memory is set to **AUTOMATIC** (which is the default), its size is computed by DB2 taking into account the values specified for heaps such as **dbheap**, **locklist**, and **pckcachesz**.

Many of these component heaps are allocated incrementally, up to the limit specified, and this may lead you to believe that over-configuring these component heaps may not be a problem. This is *not* the case, since the component heaps' upper limit specifications impact the total memory allocated and result in allocation failures, as well as paging space consumption. problems.

In deferred memory allocation schemes (as in UNIX), memory allocation does not result in paging or swap space backing until the memory is actually used. Refer to "AIX virtual memory architecture review" on page 335 for further details on paging in AIX.

For recommendations and monitoring guidelines on these heaps, refer to 3.4, "System environment considerations" on page 198.

Database manager shared memory

Also known as instance shared memory, this memory is used by DB2 to manage the activities of all connections for all databases associated with this instance. This includes the following:

- **Monitor heap** holds Database System Monitor data. The database manager configuration parameter **mon_heap_sz** specifies the number of 4 K pages to hold this data.
- **Audit buffer size** holds audit data if auditing is enabled. The database manager configuration parameter **audit_buf_sz** specifies the number of 4 K pages to hold this data.

  The writing of the audit records may occur synchronously or asynchronously.

  – When **audit_buf_sz** is zero (default) audit records are written synchronously, the event generating the audit record waits until the record is written to the disk.

– When `audit_buf_sz` is greater than zero, the audit record is written asynchronously and the event generating the audit record does not wait until the record is written to disk.

Note: In the asynchronous case, it is possible for audit records to remain in an unfilled buffer for a long time. To prevent this from happening, the database manager forces the writing of the audit records on a regular basis. An authorized user of the audit facility may also flush the audit buffer by using the command **`db2audit flush`**.

All EDUs in a partition[2] are attached to this memory.

The **`instance_memory`** database manager configuration specifies the amount of memory in 4 K pages which should be reserved for instance management; this includes memory areas that describe the databases on the instance. If this parameter is set to **`AUTOMATIC,`** DB2 calculates the amount of memory needed for the current configuration.

There are no overflows into or out of this heap.

Database manager shared memory is allocated *in full* at instance start time (**`db2start`**) and deallocated when it is stopped (**`db2stop`**).

The memory visualizer described in "Memory Visualizer" on page 103 gives information about allocation and utilization at the database manager shared memory level, but not at the granularity of all the individual heaps comprising the database manager shared memory.

Table 2-4 summarizes the main features of database manager shared memory heaps.

Table 2-4 Database manager shared memory heaps

Parameter	**Memory allocation usage**	**Default value 4 KB units**	**Allocated at**	**Deallocated at**	**Configurable online?**
instance_memory	► Full allocation ► Hard limit	Automatic	db2start	db2stop	No
mon_heap_sz	► Full allocation ► Hard limit	56 (UNIX) 12 (Windows)	db2start	db2stop	No

[2] A partition is a part of a database that consists of its own data, indexes, configuration files, and transaction logs. A single partition database is a database having only one database partition, and all the data in the database is stored in that partition. A partitioned database is a database with two or more database partitions, and tables can be located in one or more database partitions.

Parameter	Memory allocation usage	Default value 4 KB units	Allocated at	Deallocated at	Configurable online?
audit_buf_sz	► Full allocation ► Hard limit	0	db2start	db2stop	No

Database shared memory

Also known as database global memory, this memory is used by DB2 to manage the activities of all connections to a specific database associated with this instance. This includes the following:

- **Buffer pools** hold table and index pages when they are read from disk. Their size is specified in the CREATE/ALTER BUFFERPOOL statement.
- **Control blocks for buffer pool descriptors**, about one page for every 30 pages in the buffer pool.
- **Database heap** (database configuration parameter **`dbheap`**) holds space for many items including:
  - Temporary memory for utilities
  - Event Monitor buffers
  - Log buffers as specified by the database configuration parameter **`logbufsz`**
  - Temporary overflows from package cache (database configuration parameter **`pckcachesz`**)
  - Temporary overflows from catalog cache (database configuration parameter **`catalogcache_sz`**)

  Note: The buffer pool descriptors, including page descriptors, have no overhead in the database heap; they comes out of the database shared memory and are sized automatically.

- **Utility heap** specifies the maximum amount of memory (database configuration parameter **`util_heap_sz`**) that can be used for utilities such as **`backup`**, **`restore`** and **`load`** (including load recovery).

  This area of memory may also contain temporary overflows from the package cache (database configuration parameter **`pckcachesz`**) and from the catalog cache (database configuration parameter **`catalogcache_sz`**).

- **Lock list** specifies the amount of memory (database configuration parameter **`locklist`**) allocated to store all locks held by all applications concurrently connected to this database. The total amount of locks that can used by all concurrent applications per each database.

- **Catalog cache** specifies the amount of memory (database configuration parameter `catalogcache_sz`) used to cache system catalog information.This is a soft limit and can overflow to other database shared memory heaps such as `dbheap, util_heap_sz` and `pckcachesz.` This can also have overflows from the package cache (database configuration parameter `pckcachesz`).
- **Package cache** specifies the amount of memory (database configuration parameter `pckcachesz`) used to cache sections for both static and dynamic SQL statements. This is a soft limit and can overflow to other heaps such as `dbheap, util_heap_sz,` and `catalogcache_sz.`This can also have overflows from the catalog cache (database configuration parameter `catalogcache_sz`).
- **Shared sort memory** specifies an upper limit on the amount of database shared memory (database configuration parameter `sheapthres_shr`) that can be used for shared sorts at any one time. This is a hard limit which, when reached, will cause subsequent shared sorts to fail with an SQL0955C code.

  This parameter is only valid when either the database manager configuration parameter `INTRA_PARALLEL` is set or connection concentrator is enabled (database manager configuration parameters `MAX_CONNECTIONS` is greater than `MAX_COORDAGENTS`).
- **Extended storage cache**[3] can be used in systems with very large memory (typically greater than 4 GB); what this permits is the use of this extra region of memory as an extended buffer pool. This extended storage acts as an extended look-aside buffer for the main buffer pools.

  The size of this extended memory is defined in terms of the number and size memory segments as specified by the database configuration parameters `estore_seg_sz` and `num_estore_segs`.

All EDUs working on behalf of a database are attached to this memory.

The `database_memory` database configuration parameter specifies the minimum amount of memory in 4 K pages to be reserved for a given database; it does not include the database manager shared memory or the application group memory. It can be set to `AUTOMATIC`, which lets DB2 calculate the memory needed for the current configuration.

Database shared memory is allocated in full at the first connection to the database or when the database is activated, and deallocated when the database is deactivated.

The memory visualizer, described in "Memory Visualizer" on page 103, gives information about allocation and utilization at the database shared memory level

[3] This is not required in 64-bit systems, as all memory is directly accessible by applications in such cases.

but not at the granularity of all the individual heaps comprising the database shared memory.

Table 2-5 summarizes the main features of database global memory heaps.

Table 2-5 Database shared memory heaps

Parameter	Memory allocation usage	Default value 4 KB units	Allocated at	Deallocated at	Configurable online?
database_memory	► Full allocation ► Hard limit	Automatic	First connection or Activate database	Deactivate database	No
SIZE in CREATE/ALTER BUFFERPOOL statement	► Full allocation ► Hard limit	56 (UNIX) 12 (Windows)	First connection or Activate database	Deactivate database	Yes
dbheap	► Minimum for DB2 to get started, then incremental to maximum	Automatic	First connection or Activate database	Deactivate database	Yes
util_heap_sz	► Incremental allocation ► Hard limit	5000	As required by the utility	When no longer need by the utility	Yes
catalogcache_sz	► Incremental allocation ► Soft limit	-1	First connection or Activate database	Deactivate database	Yes

Parameter	Memory allocation usage	Default value 4 KB units	Allocated at	Deallocated at	Configurable online?
pckcachesz	► Incremental allocation ► Soft limit	-1	First connection or Activate database	Deactivate database	Yes
locklist	► Full allocation ► Hard limit	100 (UNIX) 50 (Windows)	First connection or Activate database	Connect reset (when there are no other agents connected at the time) or Deactivate database	Yes
sheapthres_shr	► Incremental allocation ► Hard limit	sheapthres	When sort initiated	When sort completed	No
estore_seg_sz num_estore_segs	► Full allocation ► Hard limit	16000 and 0 respectively	First connection or Activate database	Deactivate database	No

(Application Group)/Application Shared Memory

Application Group Shared Memory and Application Shared Memory are used by all agents (both coordinating and subagents) that work for an application. This memory is also used to store descriptor information for declared temporary tables.

This memory is *only* allocated when either the database manager configuration parameter **`INTRA_PARALLEL`** is set, or connection concentrator is enabled (database manager configuration parameters **`MAX_CONNECTIONS`** is greater than **`MAX_COORDAGENTS`**). The following database configuration parameters specify the memory utilization of this area.

- **`appgroup_mem_sz`** determines the size of the application group shared memory segment, and it stores information that needs to be shared between agents working on the same application.

Note: Application groups help spread out access to critical resources such as shared SQL work spaces across agents. This can help reduce contention in very high user population environments. There are typically between 100 and 200 applications per application group. Within an application group shared memory segment, each application has its own application control heap, and all applications share one application group shared heap. When a system starts up there is only one application group, and as more connections are added, DB2 creates additional application groups—this is done automatically and transparently by DB2.

- **`groupheap_ratio`** specifies the percentage of memory in the application group shared memory that is devoted to the application group shared heap; the rest is the application control heap available for use by all the applications in the application group.
- **`appl_ctl_heap_sz`** specifies the size of the shared memory area allocated for each application in the application group; this area of memory is also called Application Shared Memory.

All EDUs working on behalf of a particular application are attached to an Application Shared Memory region for that application. In addition, all EDUs working on behalf of a particular application are attached to the Application Group Shared Memory region for the application group that application is a member of.

There are no overflows into or out of this heap.

This memory is allocated incrementally when an application starts, and is deallocated when the application completes.

The memory visualizer described in “Memory Visualizer” on page 103 gives information about allocation and utilization at the application shared memory level, but not at the granularity of all the individual heaps comprising the application shared memory.

Table 2-6 on page 40 summarizes the main features of application group shared memory and application shared memory heaps.

Table 2-6 App. group shared memory and app. shared memory heaps

Parameter	Memory allocation usage	Default value 4 KB units	Allocated at	Deallocated at	Configurable online?
appgroup_mem_sz	► Incremental allocation ► Soft limit	20000 (UNIX with local clients other than HP-UX) 10000 (Windows) 30000 for database servers with local and remote clients (HP-UX)			No
groupheap_ratio	not applicable	70	not applicable	not applicable	No
appl_ctl_heap_sz	► Incremental allocation ► Soft limit	64 for database server with local clients 128 for database server with local + remote clients	When application starts	When application completes	No

Agent private memory

Agent private memory consists of a number of individual heaps, as shown in Figure 2-7 on page 32. This memory is used manage a number of activities such as compiling, sorting and servicing of UDFs and stored procedures. Each agent may allocate and use one or more of the following heaps in agent private memory:

- **Sort heap** (database configuration parameter **`sortheap`**) defines the maximum number of private memory pages to be used for each private sort, or the maximum number of shared memory pages used for each shared sort.

Note: 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 database shared memory.

- **Application heap** (database configuration parameter **`applheapsz`**) defines the number of pages for processing SQL statements and stores copies of executing sections of SQL statements.
- **Statement heap** (database configuration parameter **`stmtheap`**) specifies the amount of memory to be used as a work space for the SQL compiler during compilation of an SQL statement.
- **Statistics heap** (database configuration parameter **`stat_heap_sz`**) specifies the maximum amount of memory used for collecting statistics using the **`runstats`** command.
- **Query heap** (database manager configuration parameter **`query_heap_sz`**) specifies the maximum amount of memory used for storing information for each query such as SQLDA, statement text, SQLCA, package name, creator, section number and consistency token.
- **Java™ heap** (database manager configuration parameter **`java_heap_sz`**) specifies the maximum amount of memory used by the Java interpreter to service Java DB2 stored procedures and UDFs.
- **Agent stack** (database manager configuration parameter **`agent_stack_sz`**) specifies the initial committed amount of memory allocated for each agent in a Windows environment. It does *not* apply to UNIX platforms.
- **Client I/O block** (database manager configuration parameter **`rqrioblk`**) specifies the size of the communication buffer between *remote applications* and database agents. When a database client requests a connection to a remote database, this communication buffer is allocated on the client.

  On the database server, a communication buffer of 32767 bytes is initially allocated, until a connection is established and the server can determine the value of **`rqrioblk`** at the client; once the server knows this value, it reallocates its communication buffer to the minimum of the client and server settings for this parameter. In addition to this communication buffer, **`rqrioblk`** is also used to determine the I/O block size at the database client when a blocking cursor is opened.

Each EDU has its own private memory.

Different components of agent private memory are allocated on a demand basis, and deallocated as soon as their task is complete. Agent private memory is allocated for an agent when the agent is assigned as the result of a connect request, and deallocated when the agent terminates.

There are no overflows into or out of this heap.

The memory visualizer described in "Memory Visualizer" on page 103 gives information about allocation and utilization at the agent private memory level, but not at the granularity of all the individual heaps comprising the agent private memory.

Table 2-7 summarizes the main features of agent private memory heaps.

Table 2-7 Agent private memory heaps

Parameter	**Memory allocation usage**	**Default value 4 KB units**	**Allocated at**	**Deallocated at**	**Configurable online?**
sortheap	Private sorts ► Incremental allocation ► Soft limit Shared sorts ► Incremental allocation ► Hard limit	256	When sort is needed	When sort completes	No
applheapsz	► Incremental allocation ► Soft limit	256	When the agent is initialized to do work for an application	When the agent completes the work to be done for an application	No
stmtheap	► Full allocation ► Hard limit	2048	During precompiling or binding of an SQL statement	When precompiling or binding of an SQL statement completes	No
stat_heap_sz	► Full allocation ► Hard limit	4384	When `runstats` utility is started	When `runstats` utility completes	No
query_heap_sz	► Full allocation ► Hard limit	1000	When an application connects to a database	When an application disconnects from the database, or detaches from an instance	No

Parameter	Memory allocation usage	Default value 4 KB units	Allocated at	Deallocated at	Configurable online?
java_heap_sz	► Full allocation ► Hard limit	512	When a Java stored procedure or UDF starts	When the db2fmp process (fenced) or the db2agent process (trusted) terminates	No
agent_stack_sz	► Initial allocation size ► Upper limit depends on default reserve stack size	16	When the agent is initialized to do work for an application	When the agent completes the work to be done for an application	No
rqrioblk (remote applications)	► Full allocation ► Hard limit	32767 bytes	► 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 the remote client application disconnects from the server database ► When the blocking cursor is closed.	No

Agent/Application Shared Memory

This memory (database manager configuration parameter `aslheapsz`) is attached by coordinating agents servicing local applications, and is used for SQL request/response communications. It represents a communication buffer between the local applications and its associated agent, and is allocated as shared memory by each database agent that is started.

In addition to this communication buffer, it is also used for two other purposes as follows:

- To determine the I/O block size when a blocking cursor is opened.
- To determine the communication size between agents and db2fmp processes, which may be a user-defined function or a fenced stored procedure.

aslheapsz determines the initial size of the query heap for both local and remote clients, with the maximum size being defined by the **query_heap_sz** database manager configuration parameter.

Also included in this memory is the local applications' equivalent of **rqrioblk** as discussed in "Agent private memory" on page 40.

This memory is allocated when the database manager agent process is started for the local application, and deallocated when the database manager process is terminated.

The memory visualizer described in "Memory Visualizer" on page 103 gives information about allocation and utilization at the agent/application shared memory level, but not at the granularity of all the individual heaps comprising the agent/application shared memory.

Table 2-8 summarizes the main features of agent private memory heaps.

Table 2-8 Agent/Application Shared Memory heaps

Parameter	**Memory allocation usage**	**Default value 4 KB units**	**Allocated at**	**Deallocated at**	**Configurable online?**
aslheapsz	► Initial allocation ► Upper limit determined by the query heap size	15	When a database manager agent process is started for a local application	When the database manager agent process is terminated	No

Parameter	Memory allocation usage	Default value 4 KB units	Allocated at	Deallocated at	Configurable online?
rqrioblk (local applications)	► Full allocation ► Hard limit	32767 bytes	► When a local client application issues a connection request for a server database ► When a blocking cursor is opened, additional blocks are opened at the client	► When the local client application disconnects from the server database ► When the blocking cursor is closed.	No

2.3 Single user transaction/query flow

In this section, we describe the flow of a single user transaction involving a connect to a database, one or more SQL statements, followed by a commit and disconnect from the database. We also identify some of the key database manager and database configuration parameters that may impact the performance of the transaction.

Important: This is a simplified explanation of the flow of a transaction and its interactions with the various components of DB2. There are also many other DB2 and non-DB2 factors and parameters that affect the performance of a transaction that are not covered here.

Figure 2-8 on page 46 and Figure 2-9 on page 49 describe some of the main components involved in the execution of a transaction when connection concentrator is enabled and disabled respectively. Figure 2-9 on page 50 provides a high level comparison of the transaction flow with and without connection concentrator enabled.

For best practices related to the various database manager and database configuration parameters mentioned in this section, refer to 3.4, “System environment considerations” on page 198.

2.3.1 Transaction flow with connection concentrator enabled

Every transaction or query begins with a connect to the database, followed by one or more SQL statements, and ends with a commit and disconnect from the database.

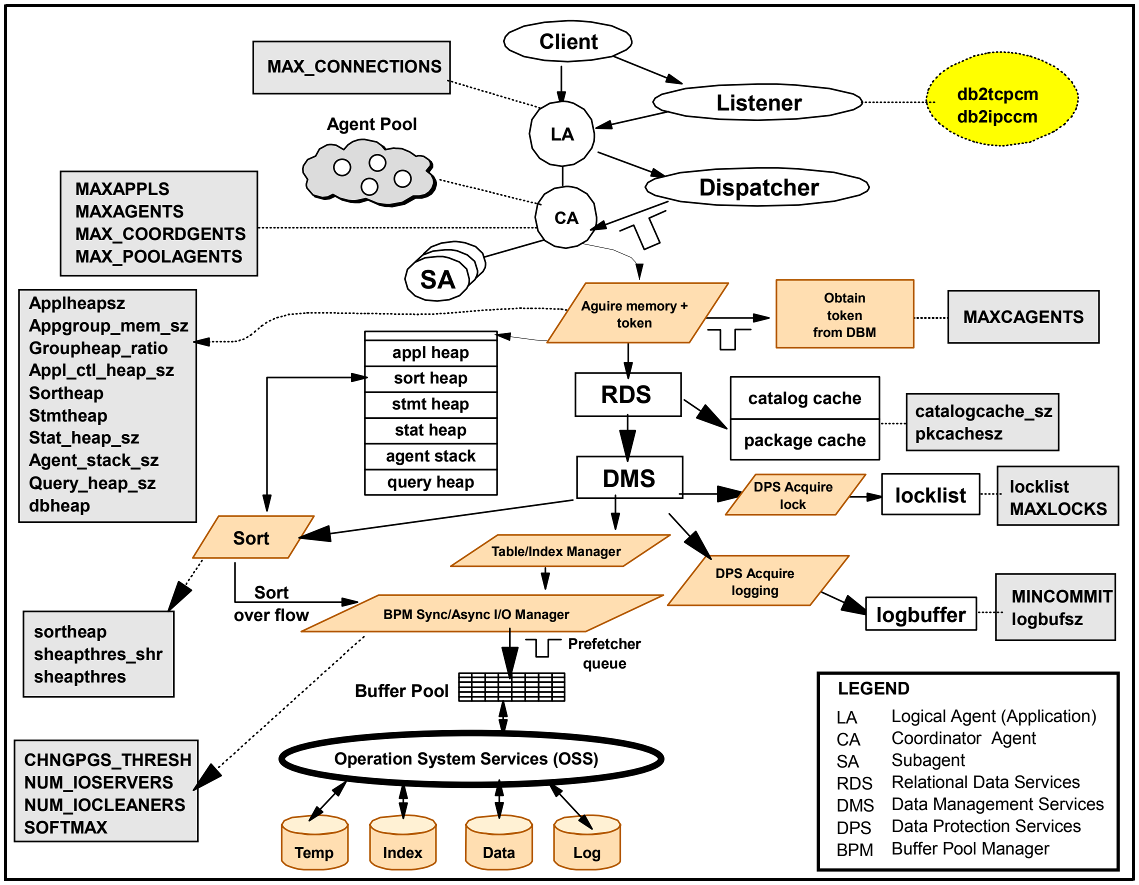


Figure 2-8 Single user transaction/query flow with CC enabled

The following series of interactions occur in DB2 as described in Figure 2-8 during the execution of a typical transaction with connection concentrator enabled:

- When the application issues a connect to a database, the appropriate DB2 listener task (db2tcpcm, or db2ipccm) creates a logical agent that is a representation of a client connection to the database, if the creation does not exceed the limit specified by the `max_connections` database manager configuration parameter.

  – If the **max_connections** limit would be exceeded by the creation of this logical agent, then an error code SQL1226N is returned to the application.

- When the first SQL call is issued by the application, the following occurs:

  a. The dispatcher (db2disp) associates a coordinator agent (if one is available) with the logical agent created and its application group; applications always belong to a particular “application group”, which is the logical representation of a set of shared resources (shared, that is, between applications within that application group).

     The coordinator agent process will then serve requests on behalf of the application for the duration of a single transaction. The coordinator agent may be created if one from the agent pool cannot be used, assuming that the creation of this coordinator agent does not exceed either the limits specified by the **maxagents**, **max_coordagents**, **maxappls** and **maxcagents** database manager and database configuration parameters.

     The coordinator agent checks the **maxcagents** database configuration parameter limit to determine whether or not a token can be acquired for the application to proceed.

     - If any of the limits (except **max_coordagents** and **maxcagents**) would be exceeded, then error code SQL1040N, or SQL1223N or SQL1226N is returned to the application.
     - If **max_coordagents** limit would be exceeded, then the request is queued until a coordinator agent becomes available.
     - If **maxcagents** limit would be exceeded, then the coordinator agent will sleep until a token becomes available, at which point the requested work will be processed.

     Note: Subagents are created if appropriate, which then execute requests on behalf of the coordinator agent.

  b. If the SQL statement is dynamic, and a reusable package is not found in the package cache (**pckcachesz**), then the SQL statement is compiled to generate an optimal access plan — the catalog cache (**catalogcache_sz**) and the statement heap (**stmtheap**) size affect the performance of this compilation process.

     The RDS component is involved in executing this function.

  c. The compiled statement is then executed.

     - Data and index access is impacted by the hit rate in the buffer pool, and this is impacted by the nature of the application workload, size of buffer pool (**SIZE**), effectiveness of prefetching (**num_ioservers**), and availability of non-dirty pages (**`num_ioservers, chngpgs_thresh, softmax`**).

    The DMS, BPM, DPS, sort, and OSS components are involved in executing this function.

  - Locks are acquired as needed and performance may be impacted by **`locklist`**, and **`maxlocks`**.

    The DPS component is involved in executing this function.

  - Sorts are performed as required, the performance of which is impacted by the sort heap (**`sortheap`**), sort heap threshold (**`sheapthres`**), and sort heap threshold for shared sorts (**`sheapthres_shr`**).

    The RDS, BPM and OSS are involved in executing this function.

  - Other heaps that impact performance are **`applheapsz, dbheap, query_heap_sz`** and **`agent_stack_sz`**.
  - Log records are written as required, the performance of which may be impacted by the size of the log buffer as controlled by **`logbufsz`**.

    The DPS component is involved in executing this function.

- Subsequent SQL calls have the same performance considerations as the first SQL call, *except* for the need to associate a coordinator agent.
- The commit call signals the end of the transaction and causes the following to occur:

  a. Force the log records to be written to disk (performance may be impacted by the **`mincommit`** parameter).

     The DPS component is involved in executing this function.

  b. Disassociate the coordinator agent from the logical agent, and return it to the agent pool, unless there is another logical agent in the queue waiting to be serviced. When pooled, the coordinator agent remains associated with the application group in question to improve performance of subsequent transactions.

- The disconnect from the database call terminates the logical agent.

2.3.2 Transaction flow with connection concentrator disabled

Figure 2-9 on page 49 describes some of the main components involved in the execution of a transaction when connection concentrator is disabled.

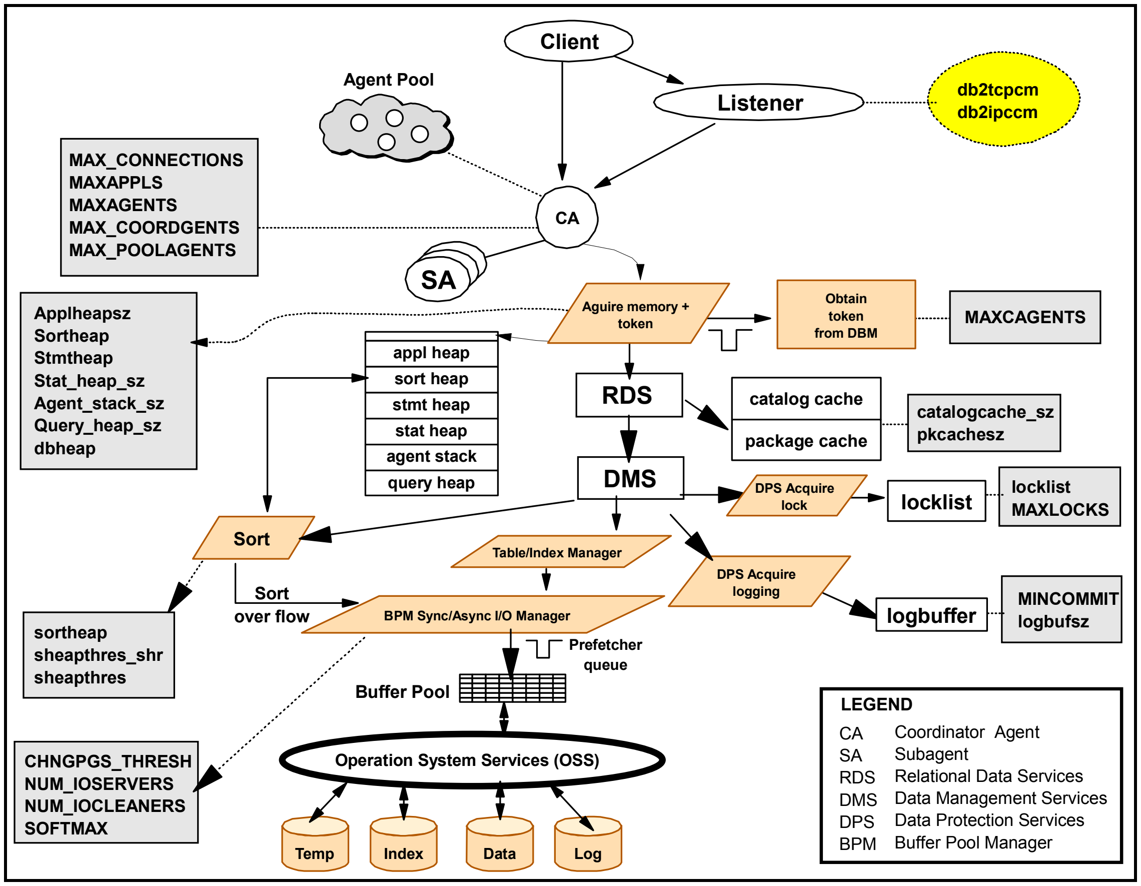


Figure 2-9 Single user transaction/query flow without connection concentrator

The series of interactions described in Figure 2-9 during the execution of a typical transaction with connection concentrator disabled *differ* from Figure 2-8 on page 46, as follows:

- There is no logical agent with connection concentrator disabled, and the coordinator agent is associated (found in the agent pool or created) during the connect to the database itself.
- There is no queuing if the **`max_coordagents`** limit is exceeded when an agent attempts to connect to a database; instead, error code SQL1226N is returned to the application.
- The coordinator agent is *not* disassociated at the end of the transaction, but at the time the agent disconnects from the database.

Table 2-9 Tuning knobs at each application call

Application calls	Connection concentrator enabled	Connection concentrator disabled
Connect to database	1. Listener checks request against MAX_CONNECTIONS limit — error code (SQL1226N) returned to application if limit exceeded 2. Creates logical agent	1. Listener checks request against MAX_CONNECTIONS, MAXAGENTS, MAX_COORDAGENTS MAXAPPLS limit — error code (SQL1040N or SQL1223N or SQL1224N or SQL1226N) returned to application if limit exceeded 2. Listener finds/creates coordinator agent (NUM_POOLAGENTS)

Application calls	Connection concentrator enabled	Connection concentrator disabled
First SQL call	1. Dispatcher checks against MAXAGENTS, MAXAPPLS limits — error code (SQL1040N, or SQL1223N or SQL1224N) returned to application if exceeded 2. Dispatcher checks MAX_COORDAGENTS, limit — queues request if limit exceeded 3. Dispatcher finds/creates coordinator agent and associates logical agent with it (NUM_POOLAGENTS) 4. Coordinator agent requests a token (MAXCAGENTS); will sleep until one becomes available 5. Begin transaction 6. Compiler compiles dynamic SQL statement (CATALOGCACHE_SZ, STMTHEAP, PCKCACHESZ) 7. Executes SQL statement (AGENT_STACK_SZ, APPLHEAPSZ, LOCKLIST, MAXLOCKS, LOGBUFSZ, QUERY_HEAP_SZ, SORTHEAP, SHEAPTHRES, SHEAPTHRES_SHR, SIZE, CHNGPGS_THRESH, NUM_IOSERVERS, NUM_IOCLEANERS, APPGROUP_MEM_SZ, GROUPHEAP_RATIO, APPL_CTL_HEAP_SZ)	1. Coordinator agent requests a token (MAXCAGENTS); will sleep until one becomes available 2. Begin transaction 3. Compiler compiles dynamic SQL statement (CATALOGCACHE_SZ, STMTHEAP, PCKCACHESZ) 4. Executes SQL statement (AGENT_STACK_SZ, APPLHEAPSZ, LOCKLIST, MAXLOCKS, LOGBUFSZ, QUERY_HEAP_SZ, SORTHEAP, SHEAPTHRES, SHEAPTHRES_SHR, SIZE, CHNGPGS_THRESH, NUM_IOSERVERS, NUM_IOCLEANERS)

Application calls	Connection concentrator enabled	Connection concentrator disabled
Subsequent SQL calls	1. Compiler compiles dynamic SQL statement (CATALOGCACHE_SZ, STMTHEAP, PCKCACHESZ) 2. Executes SQL statement (AGENT_STACK_SZ, APPLHEAPSZ, LOCKLIST, MAXLOCKS, LOGBUFSZ, QUERY_HEAP_SZ, SORTHEAP, SHEAPTHRES, SHEAPTHRES_SHR, SIZE, CHNGPGS_THRESH, NUM_IOSERVERS, NUM_IOCLEANERS, APPGROUP_MEM_SZ, GROUPHEAP_RATIO, APPL_CTL_HEAP_SZ)	1. Compiler compiles dynamic SQL statement (CATALOGCACHE_SZ, STMTHEAP, PCKCACHESZ) 2. Executes SQL statement (AGENT_STACK_SZ, APPLHEAPSZ, LOCKLIST, MAXLOCKS, LOGBUFSZ, QUERY_HEAP_SZ, SORTHEAP, SHEAPTHRES, SHEAPTHRES_SHR, SIZE, CHNGPGS_THRESH, NUM_IOSERVERS, NUM_IOCLEANERS)
Commit	1. Force log (LOGBUFSZ, MINCOMMIT) 2. Disassociate coordinator agent from logical agent (NUM_POOLAGENTS) – coordinator agent can service any other logical agent in the queue, or moved to the agent pool	Force log (LOGBUFSZ, MINCOMMIT)
Disconnect from database	Terminate logical agent	Disassociate coordinator agent from client application and return to the agent pool (NUM_POOLAGENTS)

2.4 Multi-user (concurrent) transaction/query flow

The multi-user environment tends to expose inadequately defined parameter limits such as maximum number of coordinating agents, and constrained resources such as memory heaps.

The tripping of limits or contention for memory heaps results in performance degradation. Referring to Figure 2-8 on page 46, the following considerations apply in very high user population environments:

- Inadequate limits on **max_connections**, **maxagents**, **max_coordagents**, **maxappls**, and **maxcagents** can result in application connection failures, or unacceptable waits for service from the coordinating agent.
- Insufficient application group shared memory **appgroup_mem_sz**, **groupheap_ratio**, and **appl_ctl_heap_sz** can result in inefficient use of shared memory and performance degradation.
- Insufficient **pckcachesz** can result in unnecessary compiles of dynamic SQL statements, as well as overflows into **dbheap**, and **util_heap_sz**, which can negatively impact processes that suffer from shortages of those heaps.
- Insufficient **catalogcache_sz** results in synchronous I/Os to system catalog tables, which negatively impacts the performance of dynamic SQL compiles.
- Insufficient **locklist** results in lock escalations that can result in significant performance degradation due to lock waits, timeouts, and deadlocks.
- Insufficient buffer pool size (**SIZE**) can result in synchronous I/Os, which impact application performance.
- Insufficient number of **num_ioservers** can result in prefetch requests being queued and result in synchronous I/Os; too many can also degrade performance.
- Low limits on **sortheap**, **sheapthres** and **sheapthres_shr** may result in insufficient sort heap allocation and consequent sort overflows and synchronous I/Os, which are contributors to poor performance.
- Inadequate specifications of **chngpgs_thresh, softmax** and **num_iocleaners** can result in unduly large numbers of dirty pages in the buffer pool, thereby affecting the efficiency of prefetching as well as causing expensive synchronous reads.
- Insufficient log buffer size (**logbufsz**) can result in an application waiting for log buffers, which impacts transaction performance. A high value for **mincommit** in a lightly loaded system with small/short I transactions can result in response time overhead of up to one second, and consequent reduction in transaction throughput.

In summary, insufficient memory heaps and low limits on key thresholds tend to manifest significant performance problems in heavy, multi-user workload environments and require constant monitoring and tuning to ensure business performance objectives are met.

2.5 Key performance knobs

Every organization's configuration, workload, and business application priorities tend to be unique, which correspondingly requires a custom approach to configuring, monitoring, and tuning their DB2 environment.

DB2 provides a number of tuning knobs for customizing an organization's DB2 environment in order to deliver good performance. This subsection lists and briefly describes key performance knobs available to the DBA to manage their environment. These performance knobs are:

- Database manager configuration parameters
- Database configuration parameters
- DB2 registry and environment variables

Note: Recommendations and monitoring guidelines for these parameters are discussed in Chapter 3, "Application design and system performance considerations" on page 107.

DB2 UDB version 8 provides a Configuration Advisor Wizard and an **`autoconfigure`** command described in "Configuration Advisor and AUTOCONFIGURE" on page 54 to help the DBA define optimal database manager and database configuration parameters for their environment.

The Configuration Advisor sets database manager configuration parameters in addition to database configuration parameters, and should only be run against a database that is the only database associated with that DB2 instance. The Configuration Advisor only recommends values for a little over 30 of hundreds of database manager and database configuration parameters and DB2 registry and environment variables available for configuring by the DBA.

Important: We strongly recommend the use of the Configuration Advisor or AUTOCONFIGURE command to tune your system.

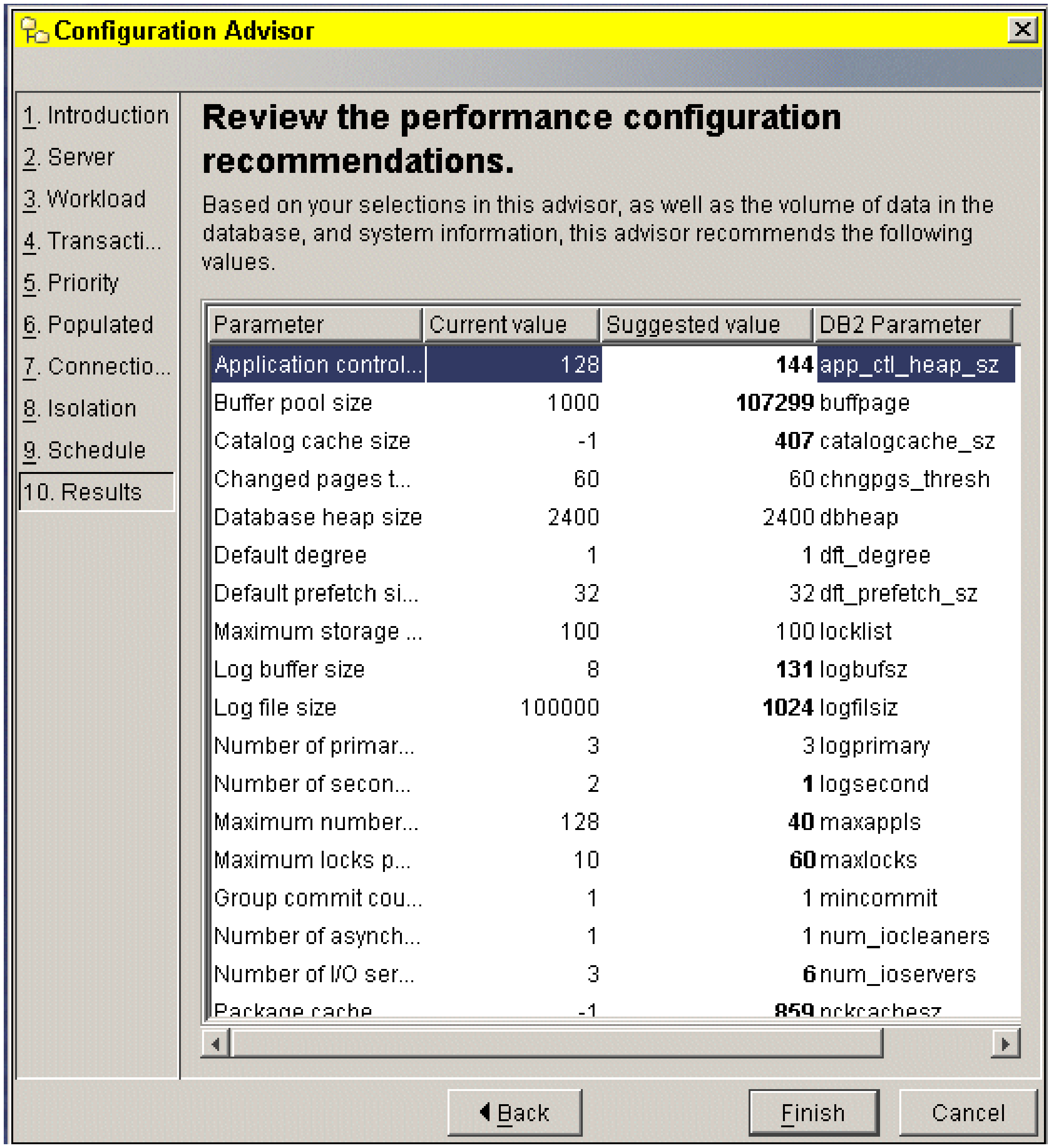
For detailed information on database manager and database configuration parameters, and DB2 registry and environment variables, refer to *DB2 UDB Administration Guide: Performance*, SC09-4821.

2.5.1 Configuration Advisor and AUTOCONFIGURE

The Configuration Advisor Wizard can be invoked from the Control Center using **Tools** -> **Wizards** -> **Configuration Advisor.** The wizard then presents a series of screens with questions regarding your environment and application workload, and generates the screen of suggested parameter settings shown in shown in Figure 2-10 on page 56.

Example 2-1 on page 57 shows the script generated for the sample database by the Configuration Advisor using most (but not all) of the default settings.

Note: Configuration Advisor actually executes on the target DB2 system, and performs auto discovery of the target DB2 environment's available resources before coming up with its recommendations for the various configuration parameters.


Configuration Advisor

1. Introduction
2. Server
3. Workload
4. Transacti...
5. Priority
6. Populated
7. Connectio...
8. Isolation
9. Schedule
10. Results

Review the performance configuration recommendations.

Based on your selections in this advisor, as well as the volume of data in the database, and system information, this advisor recommends the following values.

Parameter	Current value	Suggested value	DB2 Parameter
Application control...	128	**144**	app_ctl_heap_sz
Buffer pool size	1000	**107299**	buffpage
Catalog cache size	-1	**407**	catalogcache_sz
Changed pages t...	60	60	chngpgs_thresh
Database heap size	2400	2400	dbheap
Default degree	1	1	dft_degree
Default prefetch si...	32	32	dft_prefetch_sz
Maximum storage ...	100	100	locklist
Log buffer size	8	**131**	logbufsz
Log file size	100000	**1024**	logfilsiz
Number of primar...	3	3	logprimary
Number of secon...	2	**1**	logsecond
Maximum number...	128	**40**	maxappls
Maximum locks p...	10	**60**	maxlocks
Group commit cou...	1	1	mincommit
Number of asynch...	1	1	num_iocleaners
Number of I/O ser...	3	**6**	num_ioservers
Package cache	-1	**859**	pckcachesz

Figure 2-10 Configuration Advisor wizard

Example 2-1 Database manager and database configuration parameters set

```
UPDATE DATABASE CONFIGURATION FOR SAMPLE USING APP_CTL_HEAP_SZ 128;
UPDATE DATABASE CONFIGURATION FOR SAMPLE USING BUFFPAGE      118915;
UPDATE DATABASE CONFIGURATION FOR SAMPLE USING CATALOGCACHE_SZ 343;
UPDATE DATABASE CONFIGURATION FOR SAMPLE USING CHNGPGS_THRESH  60;
UPDATE DATABASE CONFIGURATION FOR SAMPLE USING DBHEAP        600;
UPDATE DATABASE CONFIGURATION FOR SAMPLE USING LOCKLIST       50;
UPDATE DATABASE CONFIGURATION FOR SAMPLE USING LOGBUFSZ       65;
UPDATE DATABASE CONFIGURATION FOR SAMPLE USING LOGFILSIZ     1024;
UPDATE DATABASE CONFIGURATION FOR SAMPLE USING LOGPRIMARY      3;
UPDATE DATABASE CONFIGURATION FOR SAMPLE USING LOGSECOND       0;
UPDATE DATABASE CONFIGURATION FOR SAMPLE USING MAXAPPLS       40;
UPDATE DATABASE CONFIGURATION FOR SAMPLE USING MAXLOCKS       60;
UPDATE DATABASE CONFIGURATION FOR SAMPLE USING MINCOMMIT       1;
UPDATE DATABASE CONFIGURATION FOR SAMPLE USING NUM_IOCLEANERS  1;
UPDATE DATABASE CONFIGURATION FOR SAMPLE USING NUM_IOSERVERS   4;
UPDATE DATABASE CONFIGURATION FOR SAMPLE USING PCKCACHESZ     859;
UPDATE DATABASE CONFIGURATION FOR SAMPLE USING SOFTMAX       120;
UPDATE DATABASE CONFIGURATION FOR SAMPLE USING SORTHEAP       192;
UPDATE DATABASE CONFIGURATION FOR SAMPLE USING STMTHEAP      2048;
UPDATE DATABASE CONFIGURATION FOR SAMPLE USING DFT_DEGREE      1;
UPDATE DATABASE CONFIGURATION FOR SAMPLE USING DFT_PREFETCH_SZ 32;
UPDATE DATABASE CONFIGURATION FOR SAMPLE USING UTIL_HEAP_SZ   39638;
UPDATE DATABASE MANAGER CONFIGURATION USING SHEAPTHRES       1140;
UPDATE DATABASE MANAGER CONFIGURATION USING INTRA_PARALLEL    OFF;
UPDATE DATABASE MANAGER CONFIGURATION USING MAX_QUERYDEGREE    1;
UPDATE DATABASE MANAGER CONFIGURATION USING MAXAGENTS        400;
UPDATE DATABASE MANAGER CONFIGURATION USING NUM_POOLAGENTS     400;
UPDATE DATABASE MANAGER CONFIGURATION USING NUM_INITAGENTS     0;
UPDATE DATABASE MANAGER CONFIGURATION USING FCM_NUM_BUFFERS    4096;
UPDATE DATABASE MANAGER CONFIGURATION USING FCM_NUM_RQB       0;
CONNECT TO SAMPLE;
ALTER BUFFERPOOL IBMDEFAULTBP SIZE 118915;
COMMIT;
CONNECT
RESET;
```

Note: These recommendations should be validated through actual measurements in regression test environments before committing the changes in the production environment.

The **`autoconfigure`** command delivers the same results as the Configuration Advisor wizard, by accepting the same input via options listed and described in Table 2-10 on page 58. **`autoconfigure`** calculates and displays the optimum values for the buffer pool size, database configuration, and database manager configuration parameters, with the option of applying these recommended values immediately.

The AUTOCONFIGURE option may also be used with the **`CREATE DATABASE`** command to configure databases as soon as they are created.

For more details on the **`autoconfigure`** command, refer to *DB2 UDB Command Reference*, SC09-4828.

Table 2-10 AUTOCONFIGURE command options

Option name	Valid value	Default value	Explanation
mem_percent	1 - 100	80	The percentage of real memory to dedicate to this database.
workload_type	simple, mixed, complex	mixed	Simple workloads tend to be I/O intensive and mostly transactions, whereas complex workloads tend to be CPU intensive and mostly queries.
num_stmts	1 - 1000000	10	Number of statements per unit of work.
tpm	1 - 50000	60	Number of transactions per minute.
admin_priority	performance, recovery, both	both	Optimize for better performance (more transactions per minute) or better recovery time.
is_populated	yes, no	yes	Is the database populated with data.
num_local_apps	0 - 5000	0	Number of connected local applications
num_remote_apps	0 - 5000	10	Number of connected remote applications
isolation	RR, RS, CS, UR	RR	Isolation level of applications connecting to this database
bp_resizeable	yes, no	yes	Are buffer pools resizeable?

2.5.2 Database Manager (DBM) configuration parameters

A Database Manager Configuration file (DBM) is created at the time 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 the instance. Some of the configuration parameters can be set to `AUTOMATIC`, which lets DB2 automatically adjust these parameter values to reflect current resource requirements.

There is one DBM configuration file for each client installation, which contains information about the client enabler for a specific workstation. A subset of the parameters available for a server are applicable to the client.

In UNIX environments, the DBM file is stored in a file named db2systm 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.

For some database manager configuration parameters, the database manager must be stopped (**db2stop**) and then restarted (**db2start**) in order for new parameter values to take effect; other parameters can be configured online.

Attention: Do *not* modify the db2system file using a method other than those provided by DB2; you may make the database unusable.

Table 2-11 lists key database manager configuration parameters that may have a significant impact on the performance of your specific environment.

Table 2-11 Database Manager configuration parameters

Parameter	Description
agentpri	This parameter controls the priority given to all agents and other database manager instance processes and threads by the operating system scheduler. This priority determines how CPU time is given to the DB2 processes, agents and threads relative to other processes and threads running on the machine. The default is -1, which means that no special no action is taken and the database manager is scheduled in the normal way that the operating system schedules all processes and threads.
aslheapsz	The communication buffer between the local application and its associated agent. It is also used to determine the I/O block size when a blocking cursor is opened, and determine the initial size of the query heap. If a request does not fit into the buffer, then it is split into multiple send and receive pairs.
audit_buf_sz	This parameter specifies the size of the buffer used when auditing the database. By giving a value greater than zero, the audit facility writes records to the disk asynchronously. This improves DB2 performance over leaving the parameter to the default (0), which writes records synchronously.

Parameter	Description
num_poolagents	The number of idle agents in the pool, which can be used as parallel subagents or as coordinator agents. If more agents are created than is indicated by this value, they will be terminated when they finish executing their current request, rather than be returned to the pool. When connection concentrator is disabled (`max_connections` is equal to `max_coordagents`), this parameter determines the maximum size of the idle agent pool. When connection concentrator is enabled, (`max_connections` is greater than `max_coordagents`), this parameter is 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 agent 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 pool to this parameter value. If this value is set to 0, agents will be created as needed and may be terminated when they finish executing the current request.
sheapthres	The parameter is an instance-level soft limit for private sorts, and a hard limit for shared sorts. When this limit is reached for shared sorts, no further shared sort memory requests will be allowed until the total shared sort memory consumption falls below the limit specified here.

2.5.3 Database (DB) configuration parameters

The Database (DB) configuration file is created with default parameter values when the database is created. There is one configuration file for each database, and its parameters specify, among other things, the amount of resources to be allocated to the database.

Parameters for an individual database are stored in a file named SQLDBCON, which is stored along with other control files for the database in the SQLnnnnn directory, where nnnnn is the number assigned when the database was created.

Some of the database configuration parameters are informational in nature and can only be read, while others are configurable and can be changed. Many of the database configuration parameters are configurable online, which means that changes can take effect immediately without deactivating the database. A few of the parameter changes take effect only after the database has been deactivated, which requires all applications to first be disconnected from the database.

Attention: Do *not* modify the SQLDBCON file using a method other than those provided by DB2; you may make the database unusable.

Table 2-12 lists key database configuration parameters that may have a significant impact on the performance of your specific environment.

Table 2-12 Database configuration parameters

Parameter	Description
java_heap_sz	This parameter determines the maximum size of the heap that is used by the Java interpreter to service the Java Stored procedures and UDFs. The default size is 512 by 4 K pages.
avg_appls	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. The default value is 1.
catalogcache_sz	This parameter specifies the size of the cache used to catalog information such as system tables. This caching can improve the performance of binding packages and compiling SQL statements, operations that involve checking database privileges, and operations that involve checking execute privileges. The default value is -1, which sets the catalogcache_sz to 8 x 4 K pages.
chngpgs_thresh	This parameter specifies the percentage of changed pages in the buffer pool at which the asynchronous page cleaners will be started. The default value is 60. Asynchronous page cleaners write changed pages from the buffer pool(s) to disk before the space in the buffer pool is required by database agents.
locklist	This parameter indicates the amount of storage that is allocated to the lock list, which contains the locks held by all applications concurrently connected to the database. When this limit is exceeded, lock escalation occurs which may have a significant negative performance impact as a result of lock waits, timeouts, and deadlocks.
logbufsz	This parameter specifies the amount of memory to use as a buffer for log records before writing these records to disk. By increasing the value of this parameter, the log records may be written to the disk less frequently.

Parameter	Description
maxappls	This parameter specifies the maximum number of concurrent applications that can be connected (both local and remote) to a database. Since each application that attaches to a database causes some private memory to be allocated, allowing a larger number of concurrent applications will potentially use more memory. The default is AUTOMATIC.
num_iocleaners	This parameter specifies 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 do not have to wait for changed pages to be written out to the disk. The default value is 1.
num_ioservers	This parameter specifies the number of asynchronous prefetchers for a database.These prefetchers perform prefetch and other asynchronous I/O, such as backup and restore. The default value is 3.
pckcachesz	This parameter specifies the size of the cache used to keep the sections for static and dynamic SQL statements in database shared memory. Caching the packages will reduce access to the system catalogs for reloading a package, or eliminate the need for compilation for dynamic SQL statements. The default value is -1.
seqdetect	This parameter may be used to control whether the database manager should perform sequential detection, which is the process of activating I/O prefetching by monitoring I/O. The default value is YES.
sheapthres_shr	This parameter is a hard limit on the total amount of database shared memory that can be used to sort at any one time. When the limit is reached, subsequent sorts will fail (SQL0995C). The value of 0 means that `sheapthres_shr` is equal to the `sheapthres` DBM configuration parameter. This parameter applies only when INTRA_PARALLEL is enabled, or if the connection concentrator is enabled.
sortheap	This parameter specifies the maximum number of agent private memory pages to be used for private sorts, or the maximum number of shared memory pages to be used for shared sorts. Each sort has a separate sort heap that is allocated as needed by the database manager. The sort heap is where the data is sorted. The default is 256 x 4 K pages.

Parameter	Description
softmax	This parameter is used to influence the number of log files that need to be recovered following a crash, and determine the frequency of soft checkpoints. To influence the number of log files 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. The default is 100, which tries to keep the number of logs that need to be recovered to 1.

2.5.4 DB2 registry and environment variables

In addition to the database manager and database configuration parameters, some DB2 registry and environment variables may also have a significant impact on performance. Registry and environment variables can be updated using the DB2 Configuration Assistant or the **`db2set`** command. The registry and environment variables are categorized as:

- General registry variables
- System environment variables
- Communication variables
- MPP configuration variables
- SQL compiler variables
- Performance variables
- Data link variables
- Miscellaneous variables

The command **`db2set -lr`** lists all available registry variables, some of which may or may not be useful or supported in your environment.

DB2 configures its operating parameters by checking for registry and environment variables and resolving them in the following order:

- Environment variables are modified via the **`set`** command on Windows, or the **`export`** command on UNIX.
- Registry values are set with the **`db2set`** command in the instance level profile. This profile contains instance level settings and overrides values set at the global level.
- Registry values are set with the **`db2set`** command at the global level. This profile contains machine-wide variable settings. Any variable not defined in the node or instance levels will be evaluated at this level.

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

Table 2-13 lists some of the performance-related environment and registry variables of interest on all platforms.

Table 2-13 Registry and environment performance variables for all platforms

Variable name	Description
DB2_AVOID_PREFETCH	Specifies whether prefetch should be used during crash recovery. When set to OFF (default), prefetch is used.
DB2_BINSORT	Enables a new sort algorithm that reduces the CPU, sort and elapsed time. This new algorithm extends the extremely efficient integer sorting technique of all data types such as BIGINT, CHAR, VARCHAR, DECIMAL and combinations of these data types. The default is YES.
DB2_ENABLE_BUFPD	Specifies whether or not DB2 uses intermediate buffering to improve query performance. This buffering may not improve performance in all environments. Testing should be done to determine individual query performance improvements The default is OFF.
DB2MAXFSCRSEARCH	Specifies the number of free-space control records to search when adding a new record to a table. The default is 5, which searches 5 free space control records. Setting the value to -1 will force the database manager to search all free control records.
DB2_OVERRIDE_BPF	Specifies the size of the buffer pool 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 default is not set.

Variable name	Description
DB2_PARALLEL_IO	This parameter lets you specify the table spaces (identifiers) for which DB2 may use parallel I/O when reading or writing data from and to table space containers. 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, '*' When this variable is not enabled, the number of prefetcher requests is based on the number of containers in the table space. The default is null.

Table 2-14 lists the performance-related environment and registry variables for the AIX platform.

Table 2-14 Some of the performance variables for the AIX platform

Variable name	Description
DB2MEMDISCLAIM	On AIX, memory used by DB2 processes may have associated paging space that may remain reserved even when the associated memory has been freed; this depends upon AIX's virtual memory management allocation policy. This variable controls whether DB2 agents explicitly request that AIX disassociate the reserved paging space from the freed memory. The default is set to YES, which results in smaller paging space requirements and possibly less disk activity from paging.
DB2MEMMAXFREE	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	Used together 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. The default is set to ON.
DB2_MMAP_WRITE	Used in conjunction with `DB2_MMAP_READ` to allow DB2 to use mmap as an alternate method of I/O. The default is set to ON.

Table 2-15 on page 66 lists the performance-related environment and registry variables for the Windows NT/2000/2003 platforms.

Table 2-15 Performance registry variables for Windows NT/ 2000/2003 platform

Variable name	Description
DB2_AWE	Allows DB2 UDB on 32-bit platforms to allocate buffer pools that use up to 64 GB of memory if Windows 2000 is configured to support Address Windowing Extensions (AWE) buffer pools. The default value is set to null.
DB2NTMEMSIZE	Windows NT® requires that all shared memory segments be reserved at DLL initialization time in order to guarantee matching addresses across processes. This variable permits the user to override the DB2 default on Windows NT if necessary. The default value varies by memory segment. In most situations, the default is sufficient.
DB2NTNOCACHE	Specifies whether DB2 opens database files with a NOCACHE option. If set to ON, file system caching is eliminated. If set to OFF (default), 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 sort heap can be increased. One MB is reserved from system pool for every one GB in the file. This registry variable may be used to override the undocumented 192 MB limit for the cache. When this limit is reached, an out-of-resource error is given. The default is OFF.

2.6 Performance monitoring facilities

DB2 provides a number of facilities to monitor the DB2 environment. The following system and application monitoring tools are briefly described in the following sections:

- CLI/ODBC/JDBC trace
- Database System Monitor
- DB2 administration notification log
- db2batch
- db2diag.log
- DB2 Performance Expert (PE)
- Design Advisor
- Explain and Visual Explain
- Health Monitor and Health Center
- Memory Tracker
- Memory Visualizer

2.6.1 CLI/ODBC/JDBC trace

The DB2 CLI and JDBC drivers offer extensive tracing facilities. Tracing generates one or more log files when an application accesses a DB2 CLI or DB2 JDBC driver. The log files provide the following information:

- Order in which CLI or JDBC functions are called by application
- Contents of input and output parameters passed to and received from CLI or JDBC functions
- Return code and error or warning messages generated by DB2 CLI or DB2 JDBC functions

DB2 CLI and DB2 JDBC trace facilities help application developers debug programs, and also assist the DBA in tuning the application, as follows:

- A DB2 CLI trace may show a table being queried many times on a particular set of attributes, suggesting that an index corresponding to those attributes may improve application performance.
- Understand third party application software calls to the database, even though there is no flexibility to modify the SQL itself.

A DB2 CLI trace reads configuration parameters from the DB2 CLI configuration file db2cli.ini located in the sqllib directory. The contents of db2cli.ini may be viewed by issuing the following command:

```
db2 get cli cfg for section common
```

The following command updates the db2cli.ini file and enables the DB2 CLI trace facility:

```
DB2 update cli cfg for section common using trace 1
```

For more details, refer to *DB2 UDB Call Level Interface Guide and Reference, Volume 1*, SC09-4849 and *DB2 UDB Call Level Interface Guide and Reference, Volume 2*, SC09-4850.

The DB2 UDB Version 8 JDBC trace is invoked by embedding the code shown in Example 2-2 on page 67. The trace is written to driverLog.txt.

Example 2-2 Enabling JDBC trace in DB2 UDB Version 8

```
java.io.PrintWriter printWriter = null;
      try {
         printWriter =
            new java.io.PrintWriter(
               new java.io.BufferedOutputStream(
                 new java.io.FileOutputStream("c:/temp/driverLog.txt"),
                 4096),
              true);
      } catch (java.io.FileNotFoundException e) {
         java.lang.System.err.println(
            "unable to establish a print writer for trace");
         java.lang.System.err.flush();
         return;
      }

...
...
...

         ((com.ibm.db2.jcc.DB2Connection) con).setJccLogWriter(
            printWriter,
            com.ibm.db2.jcc.DB2BaseDataSource.TRACE_ALL);
```

If operating in a WebSphere® environment, add the following two properties to the datasource you want to trace as shown in Example 2-3, and then stop and start WebSphere.

Example 2-3 Adding properties to a datasource in WebSphere

```
Name        Type               Value
traceFile   java.lang.String   <path>\<filename>
traceLevel  java.lang.Integer  -1
```

2.6.2 Database System Monitor

Database monitoring is vital to the performance and health of the database environment. DB2 provides two facilities, Snapshot Monitor and Event Monitor, that collect information from the database manager, its databases, and any connected applications. This information can be used to:

- Forecast hardware requirements based on database usage patterns
- Analyze the performance of individual applications or SQL queries
- Track the usage of indexes and tables
- Pinpoint the cause of poor system performance
- Assess the impact of optimization activities, for instance, altering database manager configuration parameters, adding indexes, or modifying SQL queries

The Snapshot Monitor and Event Monitor are complementary monitoring facilities, each serving a different purpose:

- **Snapshot Monitor** captures a picture of the state of database activity at a particular point in time, at the moment the snapshot is taken.
- **Event Monitor** collects information about the database and any connected applications when specified events occur.

The Database System Monitor stores information it collects in entities called *monitor/data elements*. Each monitor element stores information regarding one specific aspect of the state of the database system. In addition, monitor elements are identified by unique names and store a certain type of information.

Monitor elements store data in the following element types:

- **Counter** counts the number of times an activity occurs. Counter values increase during monitoring. Most counter elements are resettable.
- **Gauge** indicates the current value for an item. Gauge values can go up and down depending on database activity (for example, the number of locks held). Gauge elements are not resettable.
- **Water mark** indicates the highest (maximum) or lowest (minimum) value an element has reached since monitoring was started. Water mark elements are not resettable.
- **Information** provides reference-type details of your monitoring activities. This can include items such as partition names, aliases, and path details. Information elements are not resettable.
- **Timestamp** indicates the date and time that an activity took place.

Monitor elements collect data for one or more logical data groups. A *logical data group* is a collection of monitor elements that gather database system monitoring information for a specific scope of database activity. Monitor elements are sorted in logical data groups based on the levels of information they provide.

The Database System Monitor provides multiple ways of presenting monitor data. Both the Snapshot Monitor and the Event Monitor have the option of storing monitor information in files or SQL tables, viewing it on screen by directing it to standard-out, or processing it in a client application.

Snapshot Monitor

Taking a snapshot gives information at a specific point in time; that is, a picture of the current state of activity in the database manager for a particular object or group of objects. Snapshot monitoring is useful in determining the current state of a database and its objects and applications.

The information collected is controlled by a set of monitor switches defined in the database manager configuration file; these switches and their default settings are listed and described in Table 2-16 on page 70. There is a considerable amount of basic monitoring data that is not under monitor switch control, and will always be collected regardless of switch settings. Figure 2-11 on page 72 shows the monitoring data collected in relation to monitor switch settings.

Note: Event Monitors are not affected by monitor switches in the same way as snapshot monitoring applications. When an Event Monitor is activated, it automatically turns on the instance level monitor switches required by the specific event types.

Table 2-16 Snapshot Monitor switches

Monitor switch	Default	Description
DFT_MON_BUFPOOL	OFF	Buffer pool activity information such as number of reads and writes, and time taken
DFT_MON_LOCK	OFF	Lock wait times and deadlock-related information
DFT_MON_SORT	OFF	Sorting information such as number of heaps used and sort performance
DFT_MON_STMT	OFF	SQL statement information such as start/stop time and statement identification
DFT_MON_TABLE	OFF	Table activity information such as rows read and written
DFT_MON_TIMESTAMP	ON	Times and timestamp information
DFT_MON_UOW	OFF	Unit of work information such as start/end times and completion status

Note: Snapshot Monitor overhead depends upon the monitor switch settings, but it generally has low performance overhead.

The following subsections briefly discuss:

- Setting monitor switches
- Taking a snapshot
- Using SQL snapshot functions

Setting monitor switches

Each monitoring application has its own logical view of the monitor switches and the system monitor data. Upon startup, each application inherits its monitor switch logical view settings from the **DFT_MON_** parameters in the database manager configuration file unless overridden by the application. This means that when counter element types are reset or initialized, it only affects the application that reset or initialized them. An application taking snapshots can reset its view of the counters for all databases or a specific database at any time by using the following command:

```
db2 reset monitor for database <database_alias>r
```

A monitoring application can review and alter its logical view of the monitor switch settings using the following commands:

```
db2 get monitor switches
db2 update monitor switches using sort on
```

The switches remain active until the application (CLP) detaches, or they are deactivated with another **update monitor switches** command.

Note: Changes to the switch settings at the application level only affect the application or session (CLP window) from where the switch was changed; it does not affect the database manager switch settings.

The database manager level switches may be changed by the following command:

```
db2 update dbm cfg using sort on
```

This database manager level monitor switch update requires the application performing the update to be explicitly attached to the instance for the updates to dynamically take effect. Other existing snapshot applications will not be affected by the dynamic update. New monitoring applications inherit the updated instance-level monitor switch settings.

Important: Set the database manager configuration file switches only if you want to collect data as soon as the database manager is started, as this may incur unnecessary overhead.

Otherwise, each monitoring application can set its own switches and the data it collects becomes relative to the time its switches are set.

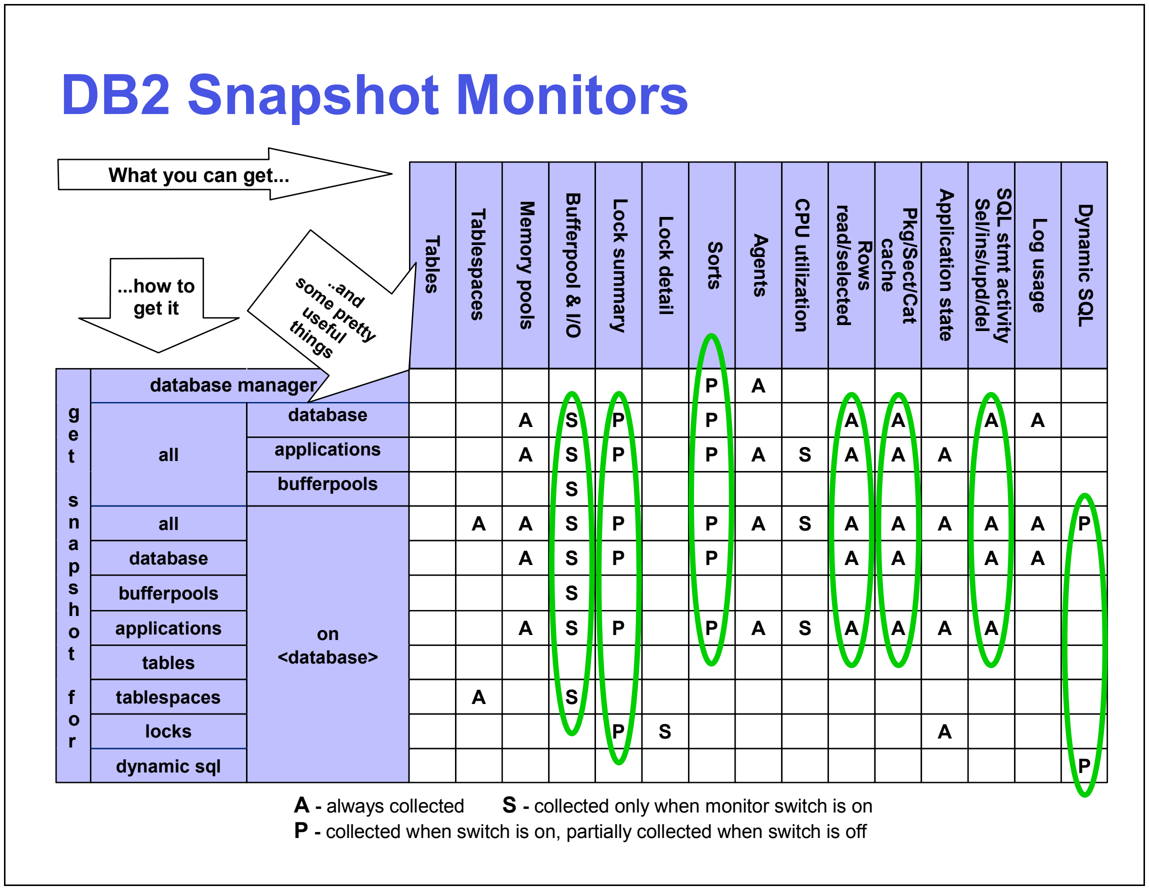
Taking a snapshot

While the Database System Monitor collects data based on monitor switch settings, the retrieval of this data is done taking a database snapshot using the **get snapshot** command.

The following command is an example of a database lock snapshot for the `sample` database:

```
db2 get snapshot for locks on sample
```

There are many options to the **get snapshot** command each of which retrieves specific information of interest. Figure 2-11 on page 72 illustrates the kinds of information retrieved for each option.


DB2 Snapshot Monitors
What you can get...
...how to get it
..and some pretty useful things

get snapshot for			Tables	Tablespaces	Memory pools	Bufferpool & I/O	Lock summary	Lock detail	Sorts	Agents	CPU utilization	Rows read/selected	Pkg/Sect/Cat cache	Application state	SQL stmt activity Sel/ins/upd/del	Log usage	Dynamic SQL
database manager									P	A							
all	database				A	S	P		P			A	A		A	A	
all	applications				A	S	P		P	A	S	A	A	A			
all	bufferpools					S											
all	on <database>			A	A	S	P		P	A	S	A	A	A	A	A	P
database					A	S	P		P			A	A		A	A	
bufferpools						S											
applications					A	S	P		P	A	S	A	A	A	A		
tables																	
tablespaces				A		S											
locks							P	S						A			
dynamic sql																	P

A - always collected **S** - collected only when monitor switch is on
P - collected when switch is on, partially collected when switch is off

Figure 2-11 DB2 Snapshot Monitor syntax and data collection

Refer to *DB2 UDB Command Reference*, SC09-4828 for details on the **get monitor**, **update monitor**, **update dbm cfg**, **get snapshot** and **reset monitor** commands mentioned here.

Using SQL snapshot functions

DB2 UDB Version 8 supports the taking of a snapshot using SQL table functions, where each snapshot table function corresponds to a particular snapshot request type. The function has as input the name of database, and the partition number.

You can use `select *` SQL statement to retrieve all monitor elements from the returned table, or select individual elements of interest.

The following is an example of selecting database manager level information using the SQL snapshot function:

```
SELECT * FROM TABLE(SNAPSHOT_DBM(-1)) AS T
```

Refer to *DB2 UDB System Monitor Guide and Reference*, SC09-4847 for complete details on SQL snapshot functions.

Event Monitor

Event Monitors are used to collect information about the database and any connected applications when specified events occur. *Events* represent transitions in database activity such as connections, deadlocks, statements and and transactions.

Attention: Whereas the Snapshot Monitor is typically used for preventative maintenance and problem analysis (routine and online/realtime event monitoring), Event Monitors are used to alert administrators to immediate problems or to track impending ones (exception monitoring).

Event Monitors have to be created for specific event types, and activated for data collection to occur. Table 2-17 lists the available event types, when the monitor data is collected, and information about each event type.

Table 2-17 Event Monitor event types

Event type	When data is collected	Available information
DEADLOCK	Detection of a deadlock	Applications involved and locks in contention.
DEADLOCK WITH DETAIL	Detection of a deadlock	Applications involved, participating statements and statement text, and a list of the locks being held.
STATEMENT	End of SQL statement	Statements start /stop time, CPU used, text of dynamic SQL, SQLCA, and other metrics such as fetch count.
TRANSACTIONS	End of unit of work	UOW start/stop time, previous UOW time, CPU consumed, locking and logging metrics.
CONNECTIONS	End of connection	All application-level counters.

Event type	When data is collected	Available information
DATABASE	Database deactivation	All database level counters.
BUFFERPOOLS	Database deactivation	Counters for buffer pool, prefetchers, page cleaner and direct I/O for each buffer pool.
TABLESPACES	Database deactivation	Counters for buffer pool, prefetchers, page cleaner and direct I/O for each table space.
TABLES	Database Deactivation	Rows read/written for each table.

Each Event Monitor has its own, private logical view of the instance's data in the monitor elements. If a particular Event Monitor is deactivated and then reactivated, its view of these counters is reset. Only the newly activated Event Monitor is affected; all other Event Monitors will continue to use their view of the counter values, plus any new additions.

Event Monitor output can be directed to SQL tables, a file, or a named pipe.

Attention: Event Monitor overhead depends upon the event types being monitored, the workload characteristics during the activation, the duration the Event Monitor is active, and the destination of the output data (to a file or table). Performance overhead may therefore be significant, and Event Monitors should be limited to short bursts with fine granularity of event types. However, when debugging is involved, it is necessary to collect enough information to diagnose the problem.

Consider using the Event Monitor in the following circumstances:

- Workload profile gathering at critical periods during the day.
- Determine transaction elapsed time, or how much CPU an SQL statement used.
- Monitor deadlocks in transactions (this may not be as important, given the advanced information now provided in the db2diag.log).
- Analyze long-running applications.

The following subsections briefly discuss:

- Creating an Event Monitor
- Activating an Event Monitor
- Performance considerations
- Viewing Event Monitor output

Creating an Event Monitor

An Event Monitor is created using the **`CREATE EVENT MONITOR`** SQL statement, which stores the metadata information in **`SYSCAT.EVENTMONITORS`**, **`SYSCAT.EVENTS`**, and **`SYSCAT.EVENTTABLES`** system catalog tables.

Event Monitors may be created via the Control Center, or by executing the following SQL statement:

```
db2 create event monitor my_event for statements write to table
```

The **`write to table`** clause specifies that the Event Monitor output should be written to SQL tables, and the database creates the necessary target tables as listed and described in Table 2-18. These tables must then be managed by the DBA, such as pruning.

Table 2-18 Write to table Event Monitor target tables

Event type	Target table names	Available information
DEADLOCKS	CONNHEADER DEADLOCK DLCONN CONTROL	Connection metadata Deadlock data Application and locks involved in deadlock Event Monitor metadata
DEADLOCK WITH DETAILS	CONNHEADER DEADLOCK DLCONN DLLOCK CONTROL	Connection metadata Deadlock data Application involved in deadlock Locks involved in deadlock Event Monitor metadata
STATEMENTS	CONNHEADER STMT SUBSECTION CONTROL	Connection metadata Statement data Statement data specific to subsection Event Monitor metadata
TRANSACTIONS	CONNHEADER XACT CONTROL	Connection metadata Transaction data Event Monitor metadata
CONNECTIONS	CONNHEADER CONN CONTROL	Connection metadata Connection data Event Monitor metadata
DATABASE	DB CONTROL	Database manager data Event Monitor metadata
BUFFERPOOLS	BUFFERPOOL	Buffer pool data Event Monitor metadata
TABLESPACES	TABLESPACE CONTROL	Table space data Event Monitor metadata

Event type	Target table names	Available information
TABLES	TABLE CONTROL	Table data Event Monitor metadata

Refer to *DB2 UDB SQL Reference Volume 2*, SC09-4845 for the syntax of the SQL statements mentioned here.

Activating an Event Monitor

Event Monitors only collect event data when they are active. To activate or deactivate an existing Event Monitor, use the **`SET EVENT MONITOR STATE`** SQL statement.

The following SQL statement activates the my_event Event Monitor:

```
db2 set event monitor my_event state 1
```

To deactivate the my_event Event Monitor, execute the following SQL statement:

```
db2 set event monitor my_event state 0
```

The status of an Event Monitor (active or inactive) can be determined by the SQL function **`EVENT_MON_STATE`** as follows:

```
select evmonname, event_mon_state(evmonname) from syscat.eventmonitors
```

A result of 1 indicates active, 0 indicates inactive, and null means undefined Event Monitor name.

Refer to *DB2 UDB SQL Reference Volume 2*, SC09-4845, for the syntax of the SQL statements mentioned here.

Performance considerations

Performance may be impacted by the following:

- Highly active Event Monitors may benefit from large buffer sizes for the target Event Monitor tables. The default is 4 pages, and can be increased as follows:

  ```
  db2 create event monitor my_event for connections, deadlock with
  details write to table buffersize 8
  ```

  Here, 8 is combined capacity (in 4 K pages) of two event table buffers. This sums to 32 K of buffer space, 16 K for each buffer.

  Note: Event Monitor buffers are allocated from `dbheap`.

- Database agents blocked by default. These wait for the event buffers to be written to the target Event Monitor tables if the buffers are full. This can degrade database performance.

  Using the NONBLOCKED option can improve performance in highly active Event Monitors, but may result in loss of event data.

  ```
  db2 create event monitor my_event for statements write to table
  buffersize 8 nonblocked
  ```

Viewing Event Monitor output

Before viewing the output, it is desirable to flush the contents of the Event Monitor buffers. This can be done by either deactivating the Event Monitor, or by using the **FLUSH EVENT MONITOR** SQL statement.

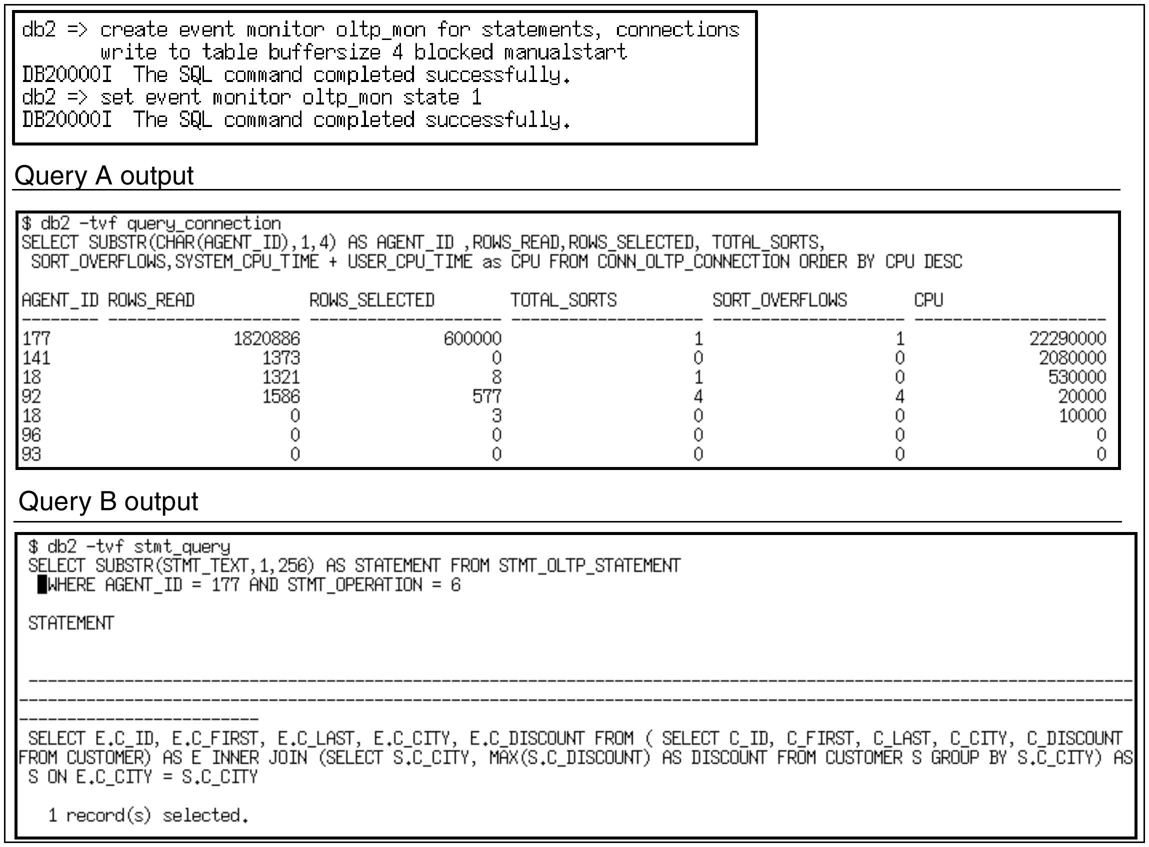
The **FLUSH EVENT MONITOR** SQL statement writes current database monitor values for all active monitor types associated with Event Monitor event-monitor-name to the Event Monitor I/O target. Therefore, a partial event record is available for Event Monitors that have a low record generation frequency (such as a database Event Monitor). Such records are noted in the Event Monitor log with a partial record identifier.

The following SQL statement flushes the Event Monitor buffers:

```
db2 flush event monitor my_event
```

The Event Monitor output data can be analyzed either by third party event analyzer GUI tools, or the **db2eva** event analyzer command.

Figure 2-12 on page 78 shows the creation and activation of an Event Monitor that is written to SQL tables, and selective querying of these tables to view the desired output.



```
db2 => create event monitor oltp_mon for statements, connections
       write to table buffersize 4 blocked manualstart
DB20000I  The SQL command completed successfully.
db2 => set event monitor oltp_mon state 1
DB20000I  The SQL command completed successfully.
```

Query A output

```
$ db2 -tvf query_connection
SELECT SUBSTR(CHAR(AGENT_ID),1,4) AS AGENT_ID ,ROWS_READ,ROWS_SELECTED, TOTAL_SORTS,
 SORT_OVERFLOWS,SYSTEM_CPU_TIME + USER_CPU_TIME as CPU FROM CONN_OLTP_CONNECTION ORDER BY CPU DESC

AGENT_ID ROWS_READ            ROWS_SELECTED        TOTAL_SORTS          SORT_OVERFLOWS       CPU
-------- -------------------- -------------------- -------------------- -------------------- --------------------
177                   1820886               600000                    1                    1             22290000
141                      1373                    0                    0                    0              2080000
18                       1321                    8                    1                    0               530000
92                       1586                  577                    4                    4                20000
18                          0                    3                    0                    0                10000
96                          0                    0                    0                    0                    0
93                          0                    0                    0                    0                    0
```

Query B output

```
 $ db2 -tvf stmt_query
 SELECT SUBSTR(STMT_TEXT,1,256) AS STATEMENT FROM STMT_OLTP_STATEMENT
  WHERE AGENT_ID = 177 AND STMT_OPERATION = 6

 STATEMENT


 ----------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------
-------------------------
 SELECT E.C_ID, E.C_FIRST, E.C_LAST, E.C_CITY, E.C_DISCOUNT FROM ( SELECT C_ID, C_FIRST, C_LAST, C_CITY, C_DISCOUNT
FROM CUSTOMER) AS E INNER JOIN (SELECT S.C_CITY, MAX(S.C_DISCOUNT) AS DISCOUNT FROM CUSTOMER S GROUP BY S.C_CITY) AS
 S ON E.C_CITY = S.C_CITY

   1 record(s) selected.
```

Figure 2-12 Viewing Event Monitor output

Query A in Figure 2-12 queries the contents of the connection Event Monitor table to find the query using the most CPU cycles. This also revealed that AGENT_ID 177 was reading a large number (1820886) of rows.

Using the agent_id (177) of the long-running query, we then queried the statement Event Monitor table using Query B to obtain the SQL statement details.

Refer to *DB2 UDB Command Reference*, SC09-4828 for details about the **db2eva** command. Refer to *DB2 UDB SQL Reference Volume 2*, SC09-4845 for the syntax of the SQL statements mentioned here. Refer to *DB2 UDB System Monitor Guide and Reference*, SC09-4847 for complete details on viewing Event Monitor output.

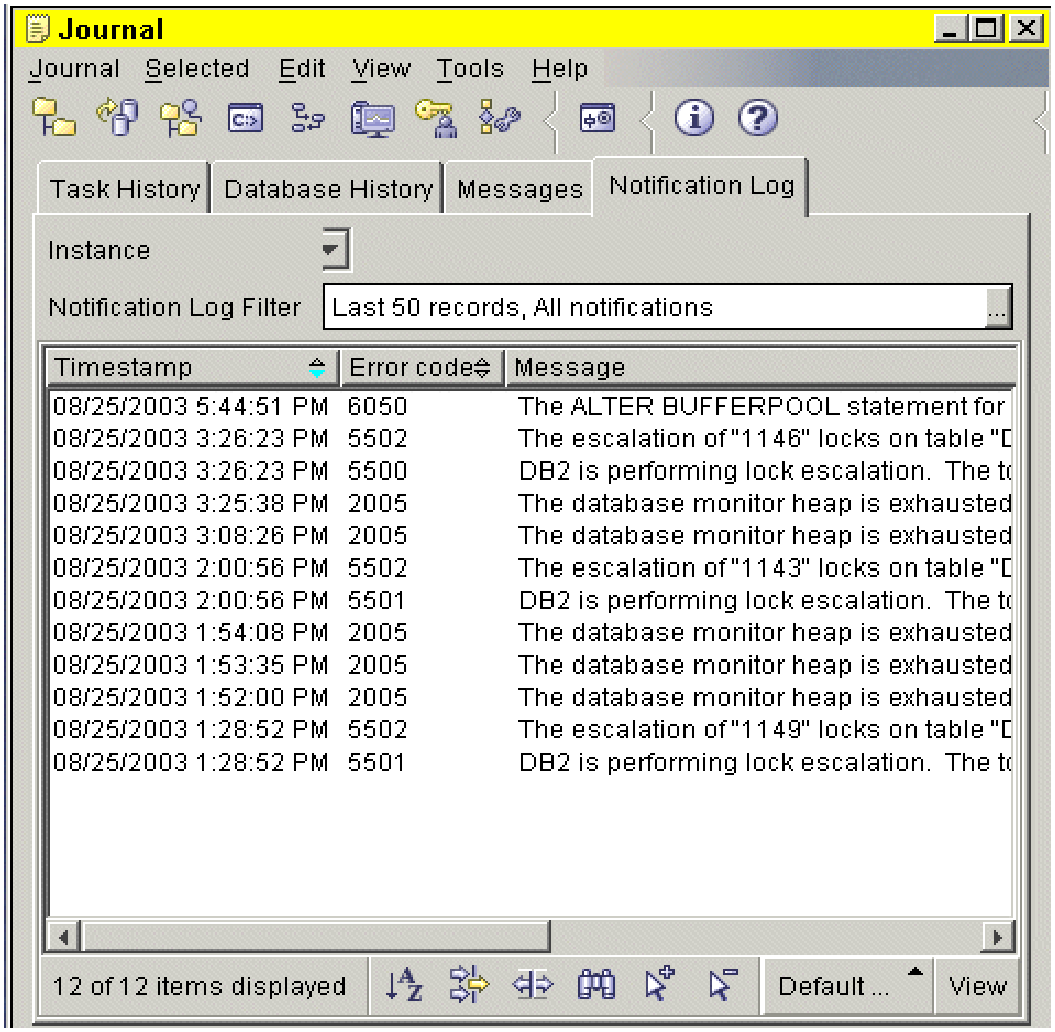
2.6.3 DB2 administration notification log

The notification log was introduced in DB2 UDB Version 8 and records a subset of the events such as error and administration logging recorded in prior DB2 versions of the db2diag.log. This information helps DBAs in problem diagnosis.

Note: In UNIX, the administration notification log is a text file called <instance>.nfy, and it is located in <INSTHOME>/sqllib/db2dump, where <INSTHOME> is the home directory of the instance owner.

On Windows, the administration notification log is written to the Windows Event Log.

The administration notification log can also be accessed from the Control Center by selecting **Tools** -> **Journal**, as shown in Figure 2-13 on page 80.



Last 50 records, All notifications

Timestamp	Error code	Message
08/25/2003 5:44:51 PM	6050	The ALTER BUFFERPOOL statement for
08/25/2003 3:26:23 PM	5502	The escalation of "1146" locks on table "D
08/25/2003 3:26:23 PM	5500	DB2 is performing lock escalation. The to
08/25/2003 3:25:38 PM	2005	The database monitor heap is exhausted
08/25/2003 3:08:26 PM	2005	The database monitor heap is exhausted
08/25/2003 2:00:56 PM	5502	The escalation of "1143" locks on table "D
08/25/2003 2:00:56 PM	5501	DB2 is performing lock escalation. The to
08/25/2003 1:54:08 PM	2005	The database monitor heap is exhausted
08/25/2003 1:53:35 PM	2005	The database monitor heap is exhausted
08/25/2003 1:52:00 PM	2005	The database monitor heap is exhausted
08/25/2003 1:28:52 PM	5502	The escalation of "1149" locks on table "D
08/25/2003 1:28:52 PM	5501	DB2 is performing lock escalation. The to

12 of 12 items displayed

Figure 2-13 The DB2 administration notification log

As in the case of the db2diag.log, the level of detailed information recorded in the administration notification log is determined by the `NOTIFYLEVEL` database manager configuration parameter, as shown in Table 2-19 on page 81.

Table 2-19 Administration notification log NOTIFYLEVEL

NOTIFYLEVEL	Description
0	No administration notification messages captured.
1	Only fatal and unrecoverable error are logged. To recover from some of these conditions, you may need assistance from DB2 service.
2	Conditions that require immediate attention from the SYSADM or DBA are logged. If the condition is not resolved, it could lead to a fatal error. This level captures Health Monitor alarms. These are in addition to NOTIFYLEVEL 1 messages.
3 (default)	Conditions that are non-threatening and do not require immediate action are logged, but it may indicate a non-optimal system. This level will capture Health Monitor alarms, warnings, and attentions. These are in addition to NOTIFYLEVEL 1 and 2 messages.
4	Informational messages as well. These are in addition to NOTIFYLEVEL 1, 2 and 3 messages.

Important: Specify `NOTIFYLEVEL` of 2 or higher for the Health Monitor to send notifications to the appropriate contacts defined in its configuration.

`NOTIFYLEVEL 4` can provide the following locking-related information:

- Locked object, lock mode, and application holding the lock.
- Current dynamic SQL statement or static package name.
- Detailed deadlock information.
- All tables for which locks are escalated, including:
  - Number of locks currently held.
  - Number of locks needed before escalation is completed.
  - Table identifier information and table name of each table being escalated.
  - Number of non-table locks currently held.
  - The table level lock to be acquired as part of the escalation—usually a Shared lock “S” or eXclusive “X” lock.

Note: `NOTIFYLEVEL 4` should normally be used only in exception monitoring situations.

For more details on the administration notification log, refer to *DB2 UDB Administration Guide: Performance*, SC09-4821.

2.6.4 db2batch

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.

This tool is particularly useful in problem diagnosis of queries that run for a very long time, since it provides detailed information about resource utilization including time spent in compilation and execution.

Example 2-4 shows an example of a db2batch input file.

Example 2-4 db2batch input file

```
--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;
```

db2batch can be invoked via the following command:

```
db2batch -d sample -f db2batch.sql
```

Example 2-5 shows the output of db2batch for the db2batch.sql file shown in Example 2-4 on page 82.

Example 2-5 db2batch sample 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
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
--#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
```

For more information on usage, syntax and options, refer to *DB2 UDB Command Reference*, SC09-4828, and to *DB2 UDB Administration Guide: Performance*, SC09-4821.

2.6.5 db2diag.log

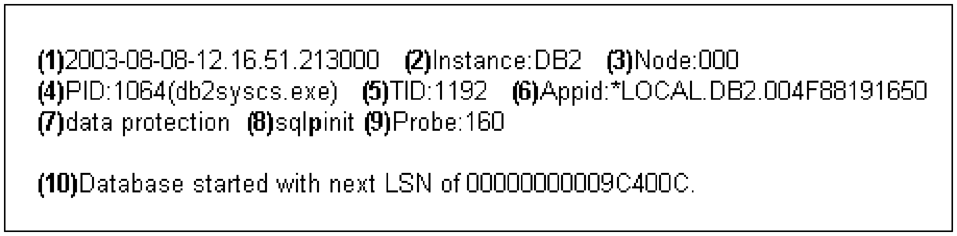
DB2 captures event information when it detects a problem in the system. The level of detailed information captured is determined by the **DIAGLEVEL** database manager configuration parameter, as shown in Table 2-20.

DB2 diagnostic log information is used for problem determination, and is primarily intended for DB2 customer support. However it may be of interest to DBAs as well. The goal is to move DBA interest information to the administration notification log.

Table 2-20 Diagnostic log DIAGLEVEL

NOTIFYLEVEL	Description
0	No diagnostic data captured
1	Severe errors only
2	All errors
3 (default)	All errors and warnings
4	All errors, warnings and informational messages

Figure 2-14 displays the header information for a sample log entry; however, not every entry will contain all of these parts.



```
(1)2003-08-08-12.16.51.213000   (2)Instance:DB2   (3)Node:000
(4)PID:1064(db2syscs.exe)   (5)TID:1192   (6)Appid:*LOCAL.DB2.004F88191650
(7)data protection   (8)sqlpinit (9)Probe:160

(10)Database started with next LSN of 00000000009C400C.
```

Figure 2-14 db2diag.log header information

The following explains the values in Figure 2-14 on page 85:

- (1) Timestamp of the message.
- (2) Name of the instance generating the message.
- (3) For multi-partition systems, the partition generating the message. In a non-partitioned database, the value is 000.
- (4) The process ID of the EDU encountering the error.
- (5) The thread ID of the EDU encountering the error (Windows only).
- (6) Identification of application associated with the process.
- (7) The DB2 component writing the message.
- (8) The name of the function generating the message. This function operates within the DB2 subcomponents writing the message. To find more about the activity performed by a function, look at the fourth letter of its name. In this example, the letter id **`p`** in the function **`sqlpinit`** indicates a data protection problem or information.

  Table 2-21 explains some of the letters in the fourth position ;the database name is also shown, when available.
- (9) The unique identifier. This number allows DB2 customer support and development to locate the pinpoint the message in the DB2 source code.
- (10) When available, a message indicating the error type and number as a hexadecimal code and a text message explaining the logged event.

Table 2-21 The function activities

The fourth letter in function activity	Description
b	Buffer pools
c	Communication between the client and server
d	Data management
e	Engine Processes
o	Operating System Calls (opening and closing files)
p	Data Protection (locking and logging)
r	Relational Data Services
s	Sorting
x	Indexing

Table 2-22 lists and explains some of the components identified in the diagnostic log.

Table 2-22 DB2 UDB - the most common engine components

Engine component	Description
`sql, squh`	DB2 backup and restore.
`sqb`	DB2 Buffer Pool Services (buffer pool, data storage management, table spaces, containers, I/O, prefetching and page cleaner).
`clp`	DB2 Command Line Processor.
`sqng, CodeGen`	This component is part of the SQL compiler and represents the last phase of statement compilation.
`sqv`	Data Services, responsible for data type comparison and conversion. This component includes routines that do the following: ► Convert between char, graphic and numeric types. ► Perform decimal, floating and integer arithmetic. ► Perform date, time, timestamp, arithmetic and conversion. ► Compare any data type to any other data type (limited to the comparisons DB2 support). These comparison routines are used by the Index Manager, Sort Services, Runtime Interpreter and other components.
`dlfm`	Data links file manager functionality which allows DB2 to manage files that are stored external to the database.
`sqd, sqdx, sqdl, dart`	DB2 Data Management Services: ► Tables, records, long field and large object columns. ► Recovery (rollforward, rollback). ► Table and record locking.
`sqp, sqdz`	Data Protection Services, logging.
`sqx`	DB2 Index Manager.
`squ, sqi, sqs, squs`	DB2 Load, Import, Export, Sort.
`sqno, sqnx, runstats`	DB2 Query Optimizer, Explain and Runstats.
`sqo, sqt, sqz`	DB2 Engine Operation System Services.

Engine component	Description
`sqe, sqkd, sqkf`	DB2 Process model, FCM, Connect, Gateway Connection Pooling, Concentrator, db2start, db2stop, create/drop at node, activate/deactivate database, add/drop node, interrupt handling, node failure and recovery, Engine Operating Services.

For more information on the db2diag.log, refer to the Online DB2 troubleshooting information via the DB2 Online Support site:

```
http://www.ibm.com/software/data/db2/udb/winos2unix/support
```

2.6.6 DB2 Performance Expert

DB2 Performance Expert is a new, IBM-supplied host-based and workstation-based performance analysis and tuning tool for both the z/OS® and multiplatform environments. Many installations have more than one DB2 system; at a minimum you might have test, production and other systems. In some cases, they also have a mix of z/OS and multiplatform environments. With DB2 PE, you can manage a heterogeneous mix of DB2 systems via a single end-user interface.

The main objective of DB2 PE is to simplify DB2 performance management. DB2 PE gives you the capability of monitoring applications, system statistics, and system parameters via a single tool.

DB2 PE integrates performance monitoring, reporting, buffer pool analysis, and a Performance Warehouse function into a single tool. It optimizes the performance of IBM DB2 for z/OS and OS/390, and of DB2 UDB, by providing a comprehensive view of DB2 performance-related information. It also provides you with reports, analysis, and recommendations.

In general, DB2 PE includes the following advanced capabilities:

- Provides detailed analysis of key performance factors that let you control and tune the performance of DB2 and DB2 applications.
- Provides a real-time online monitor, a wide range of reports, expert analysis, and an explain feature to analyze and optimize SQL statements.
- Lets you simulate certain tuning actions of buffer pools before you change your system.
- Includes a Performance Warehouse function for storing performance data and analysis functions.
- Lets you monitor Database Connection Services (DCS) connections via Performance Expert Agent.

The following subsections provide a summary of all functions. The availability of these functions, however, varies depending on whether you install DB2 Performance Expert for z/OS, DB2 Performance Expert for Multiplatforms, or one of the standalone products DB2 PM or Buffer Pool Analyzer.

Basic functions on all platforms

DB2 PE provides the following functions across all platforms:

- Analyzes and tunes the performance of DB2 and DB2 applications.
- Provides expert analysis, a real-time online monitor, and a wide range of reports for analyzing and optimizing DB2 application and SQL statements.
- Includes a Performance Warehouse for storing performance data and analysis tools.
- Defines and applies analysis functions (rules of thumb, queries) to identify performance bottlenecks.

Specific functions on z/OS

DB2 PE provides the following functions on z/OS:

- An explain feature.
- A Reporting Facility that presents detailed information about DB2 events involving CPU times, buffer pool usage, locking, I/O activity, and more.
- The ability to manage buffer pools more efficiently by providing information about current buffer pool behavior and simulating anticipated future behavior.
- Exception reports for common performance problems to help identify and quantify excessive CPU and elapsed time on a plan and package basis.

Specific functions on Multiplatforms

DB2 PE provides the following functions on Multiplatforms:

- A starter set of smart features that provide recommendations for system tuning to gain optimum throughput.
- A subset of the Reporting Facility functionality provided on z/OS.
- A subset of the Exception reports functionality provided on z/OS.
- DB2 Buffer Pool Analyzer collects data and provides reports on related event activity, to obtain information on current buffer pool behavior. It can provide these reports in the form of tables, pie charts, and diagrams.
- Monitors DB2 Connect Gateways including application and system-related information.

Attention: For a detailed description of the different capabilities and specific platform support, refer to *IBM DB2 Performance Expert for z/OS and Multiplatforms Monitoring Performance from the Workstation*, SC27-1645*,* and to the corresponding Announcement Letters.

The main components of the DB2 Performance Expert Tool are shown in Figure 2-15.

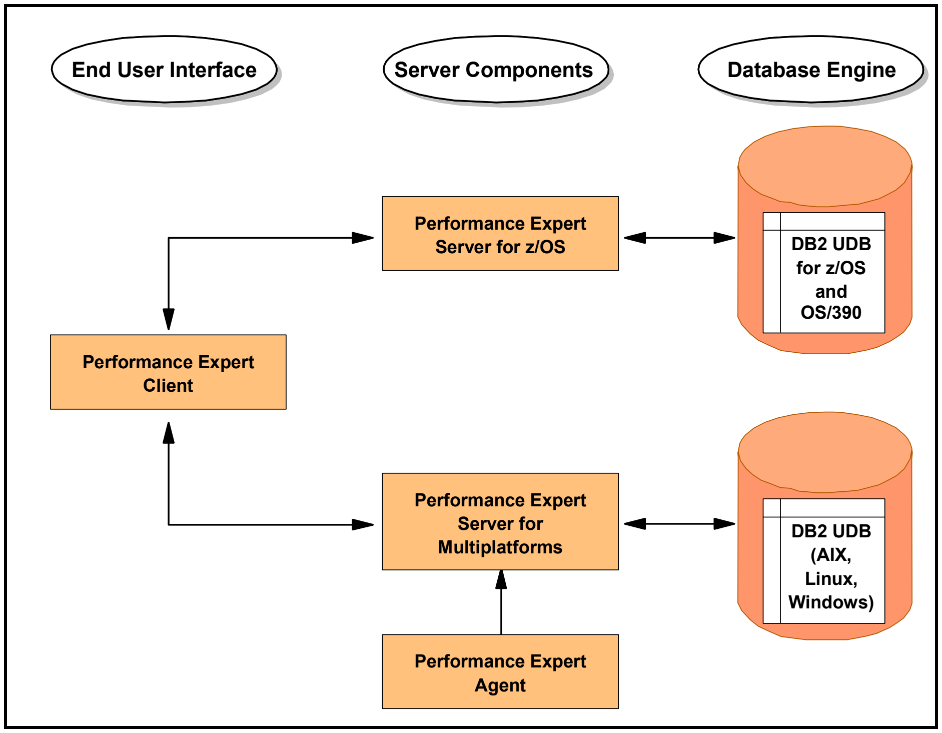

End User Interface
Server Components
Database Engine
Performance Expert Server for z/OS
DB2 UDB for z/OS and OS/390
Performance Expert Client
Performance Expert Server for Multiplatforms
DB2 UDB (AIX, Linux, Windows)
Performance Expert Agent

Figure 2-15 Main components of DB2 Performance Expert

A brief description of each of these components follows:

- Performance Expert Client designates the end-user interface of DB2 PE of z/OS and Multiplatforms.
- Performance Expert Server for z/OS accesses DB2 UDB for Z/OS and OS/390.
- Performance Expert Server for Multiplatforms accesses DB2 UDB for Windows, UNIX, Linux on zSeries®, and Linux (IA32).

- Performance Expert Agent monitors Database Connection Services (DCS) connections within the Distributed Relational Database Architecture™ (DRDA®). When Performance Expert Agent is installed on the system on which DCS connections are performed, it collects connection-related data, such as the status of a DCS connection. It also collects statistics about DB2 Connect activities. The collected data is then stored in the DB2PM database on a Performance Expert Server.

  Figure 2-16 provides an overview of how Performance Expert Agent is integrated into the system environment. The DB2PM database (also known as DB2 Performance Expert Performance Database) stores performance data collected from the initiation of various traces and Event Monitors. DB2 PE provides various canned queries to report the content of this performance data.

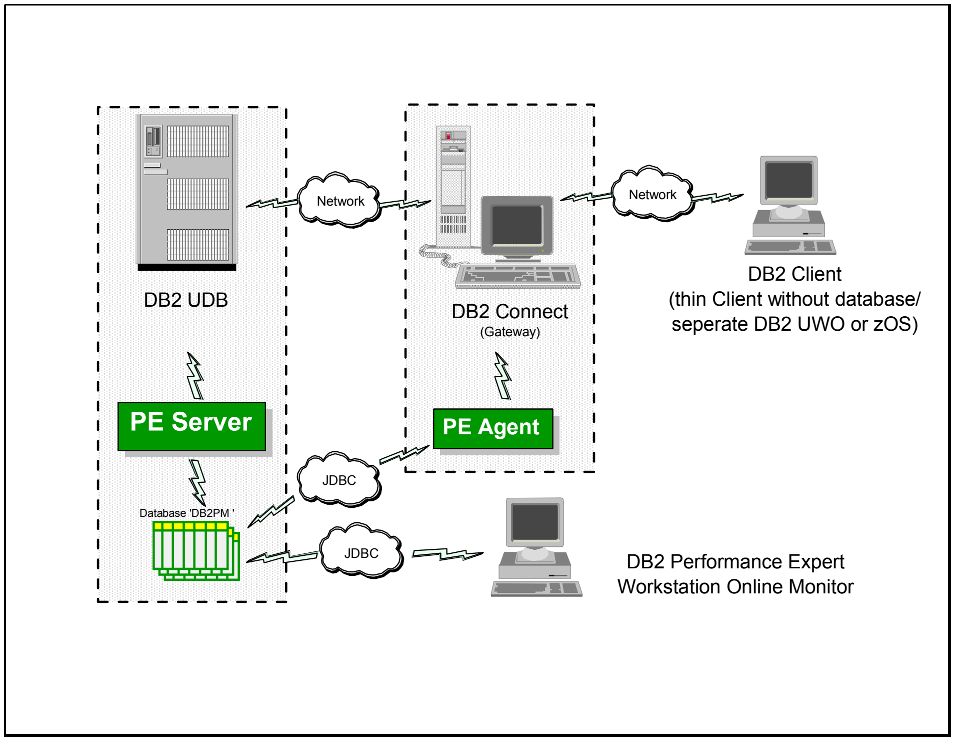


Figure 2-16 Performance Expert Agent and system environment

Figure 2-17 on page 92 describes the general environment structure of DB2 PE for Multiplatforms.

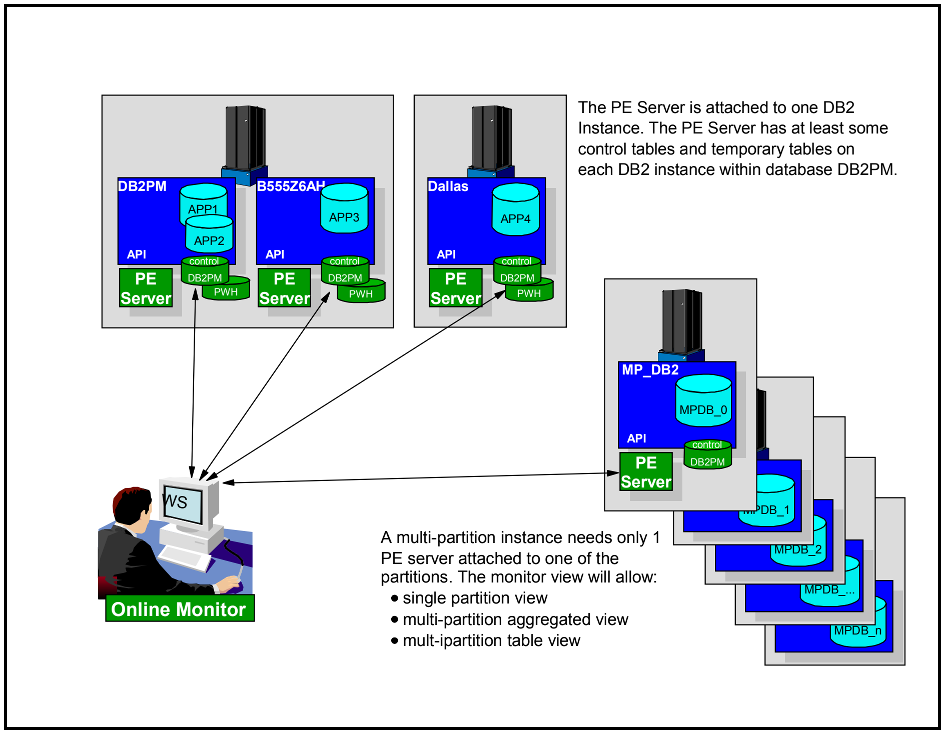

DB2PM
APP1
APP2
API
control
DB2PM
PWH
PE Server
B555Z6AH
APP3
API
control
DB2PM
PWH
PE Server
Dallas
APP4
API
control
DB2PM
PWH
PE Server
The PE Server is attached to one DB2 Instance. The PE Server has at least some control tables and temporary tables on each DB2 instance within database DB2PM.
MP_DB2
MPDB_0
API
control
DB2PM
PE Server
MPDB_1
MPDB_2
MPDB_...
MPDB_n
WS
Online Monitor
A multi-partition instance needs only 1 PE server attached to one of the partitions. The monitor view will allow:
• single partition view
• multi-partition aggregated view
• mult-ipartition table view

Figure 2-17 DB2 PE for Multiplatforms environment structure

The figure shows the DB2 PE Client monitoring two DB2 EE systems and one DB2 EEE system.

It should be noted that there is a DB2 PE Server and a DB2PM database associated with each DB2 instance, with a future goal of sharing a single DB2PM database across multiple DB2 instances.

Note: Performance Warehouse (PWH) tables are included in the DB2PM database.

Both the DB2 PE Client and the DB2 PE Server support Windows 2000, AIX, Solaris, HP-UX and Linux (IA32) platforms.

For further details, refer to *DB2 UDB Performance Expert for Multiplatforms: A Usage Guide*, SG24-6436.

2.6.7 Design Advisor

The Design Advisor helps with the design of suitable indexes for a given table by finding the best indexes for a problem query, as well as the best indexes for a set of queries that define a workload subject to optionally applied resource limits.

Note: These recommendations should be validated through actual measurements in regression test environments before committing the changes in the production environment.

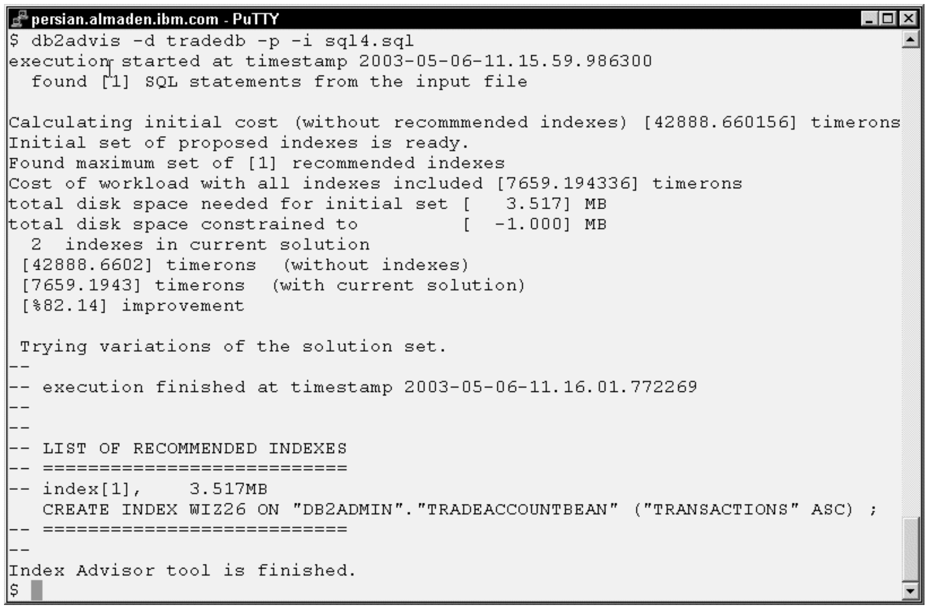
For a given workload, the Design Advisor will evaluate the existing indexes and recommend additional indexes if required.

A workload in the context of the Design Advisor 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.

The information in the workload identifies the type and frequency of the SQL statements over a given period of time. For example, your database manager may have to process 1 000 **`INSERTs`**, 10 000 **`UPDATEs`**, 10 000 **`SELECTs`**, and 1 000 **`DELETEs`** in a one month period. The Design Advisor's advising engine uses this workload information in conjunction with the database information (such as statistics) to recommend indexes. The goal of the Design Advisor's advising engine is to minimize the total workload cost.

The Design Advisor may be invoked via the **`db2advis`** command, as shown in Figure 2-18 on page 94, or from the Control Center.

.



```
$ db2advis -d tradedb -p -i sql4.sql
execution started at timestamp 2003-05-06-11.15.59.986300
  found [1] SQL statements from the input file

Calculating initial cost (without recommmended indexes) [42888.660156] timerons
Initial set of proposed indexes is ready.
Found maximum set of [1] recommended indexes
Cost of workload with all indexes included [7659.194336] timerons
total disk space needed for initial set [   3.517] MB
total disk space constrained to         [  -1.000] MB
  2  indexes in current solution
 [42888.6602] timerons  (without indexes)
 [7659.1943] timerons  (with current solution)
 [%82.14] improvement

 Trying variations of the solution set.
--
-- execution finished at timestamp 2003-05-06-11.16.01.772269
--
--
-- LIST OF RECOMMENDED INDEXES
-- ===========================
-- index[1],    3.517MB
   CREATE INDEX WIZ26 ON "DB2ADMIN"."TRADEACCOUNTBEAN" ("TRANSACTIONS" ASC) ;
-- ===========================
--
Index Advisor tool is finished.
$
```

Figure 2-18   Design Advisor

The recommended way to start Design Advisor is from the Control Center (**Tools** -> **Wizards** -> **Design Advisor**), since it can help you construct a workload. That is, it can look for recently executed SQL statements, look at recently used packages, or let you supply all the SQL statements in the workload.

For more information on the Design Advisor, refer to *DB2 UDB Administration Guide: Performance*, SC09-4821.

2.6.8 Explain and Visual Explain

DB2 provides a comprehensive explain facility that provides detailed information about the access plan that the optimizer chooses for an SQL statement. Several tools or methods give you the flexibility you need to capture, display, and analyze explain information.

Table 2-23 on page 95 lists and identifies the various tools available with the DB2 explain facility, and their characteristics.

Table 2-23 DB2 explain facility

Desired characteristics	**Visual Explain**	**db2exfmt**	**db2expln**	**dynexpln**
GUI interface	Yes			
Text output		Yes	Yes	Yes
"Quick and dirty" Static SQL analysis			Yes	
Static SQL supported	Yes	Yes	Yes	
Dynamic SQL supported	Yes	Yes	Yes	Yes*
CLI application support	Yes	Yes		
Suited for analysis of multiple statements		Yes	Yes	Yes
Detail optimizer information	Yes	Yes		

Access path information is stored in EXPLAIN tables which can be queried to retrieve the desired information. Either the GUI tool Visual Explain or the text-based `db2exfmt` tool can be used to examine the contents of the EXPLAIN tables.

\

Note: EXPLAIN tables can be created by the **`db2 -tf EXPLAIN.DDL`** command or automatically by the Control Center.

Table 2-24 lists and describes the various **EXPLAIN** tables. Figure 2-19 on page 97 shows the relationship between some of them.

Table 2-24 EXPLAIN tables

Table name	Description
EXPLAIN_INSTANCE	The main control table for all EXPLAIN information. Each row in the EXPLAIN tables is explicitly linked to one unique row in this table.
EXPLAIN_STATEMENT	This table stores the EXPLAIN snapshot, if it was requested. Data is stored as Binary Large Object (BLOB), which contains the internal representation of the access plan and decision criteria used by the DB2 optimizer. A row in the EXPLAIN_INSTANCE refers to multiple rows in this table.
EXPLAIN_OPERATOR	This table contains all the operators needed to satisfy the query. The types of operators are FETCH, GRPBY, IXSCAN, MSJOIN, NLJOIN, RIDSCN, SORT, TBSCAN, TEMP or UNIQUE.
EXPLAIN_ARGUMENT	This table contains the information for each operator. For example, for a SORT operator, arguments such as number of rows expected to be sorted are collected.
EXPLAIN_OBJECT	This table identifies the data objects required by the access plan. Types of objects are indexes, tables and table functions.
EXPLAIN_STREAM	This table represents the input and output data stream between operators and objects, for example, the number of columns represented and an estimate of the cardinality.
EXPLAIN_PREDICATE	This table identifies which predicates are applied to a specific operator.
ADVISE_WORKLOAD	This table 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.

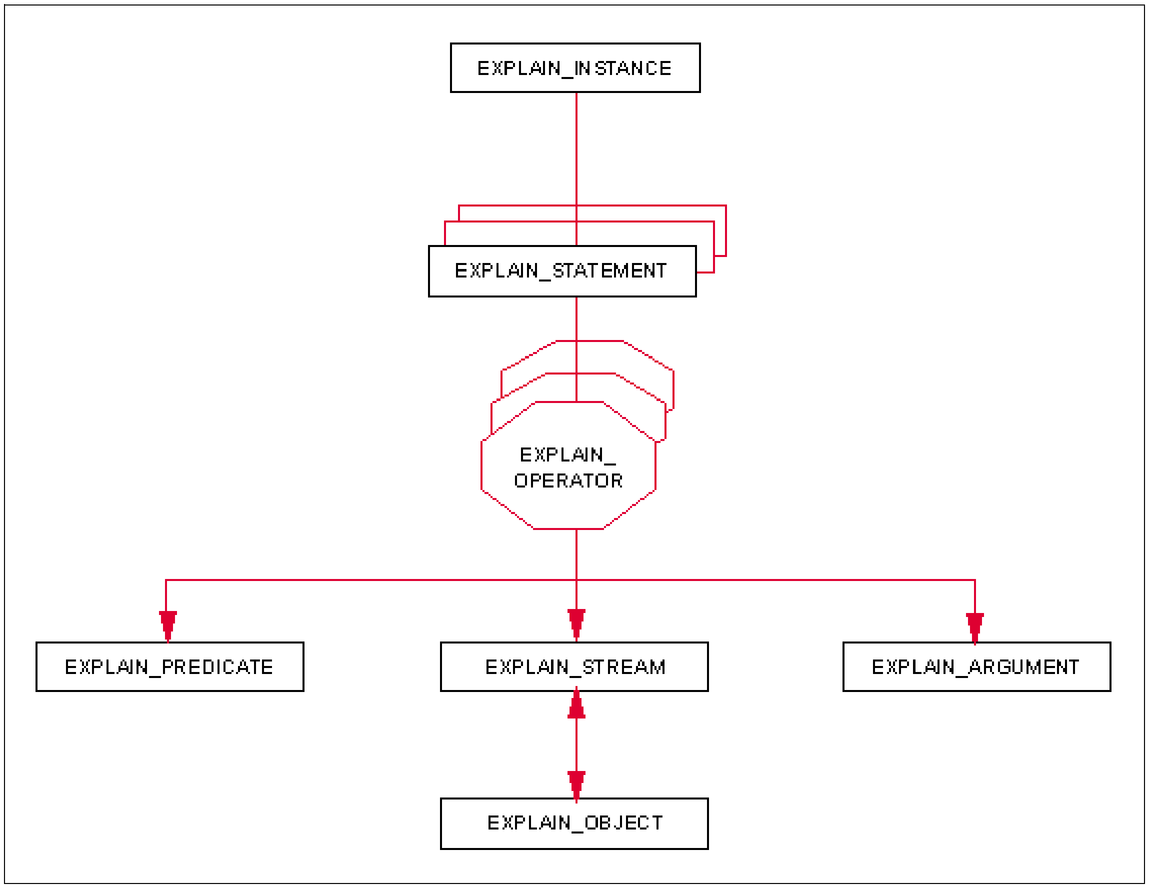
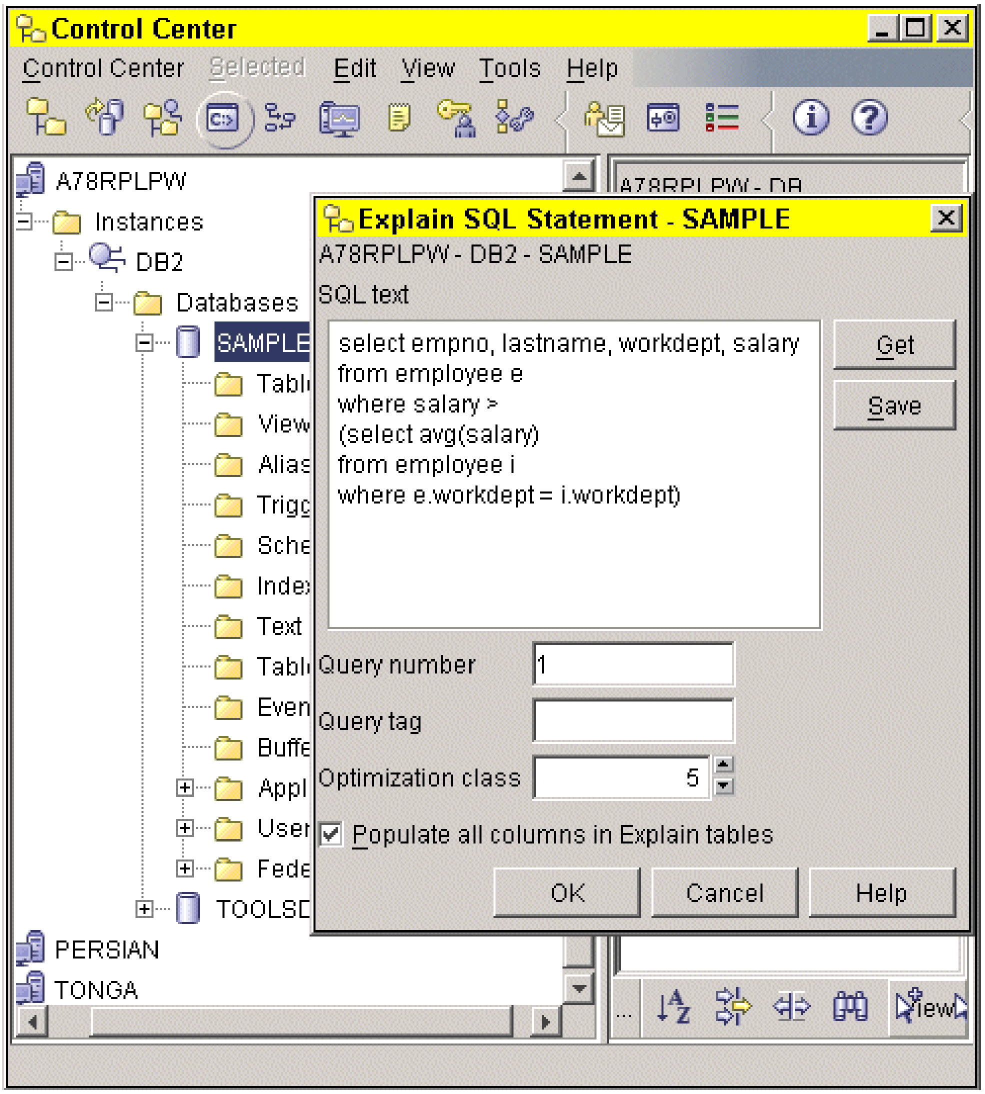


Figure 2-19 The relationship of EXPLAIN tables

Visual Explain is the GUI tool that can be invoked from the Control Center to view or print a query's access plan in the form of a graph. It can be used to view snapshots captured on other platforms for both static and dynamic SQL statements.

To explain an SQL statement by using the Visual Explain from the Control Center, right-click the database name and select **Explain SQL...**, then write the SQL to be explained, as shown in Figure 2-20 on page 98.



Explain SQL Statement - SAMPLE

A78RPLPW - DB2 - SAMPLE

SQL text

```
select empno, lastname, workdept, salary
from employee e
where salary >
(select avg(salary)
from employee i
where e.workdept = i.workdept)
```

Query number: 1

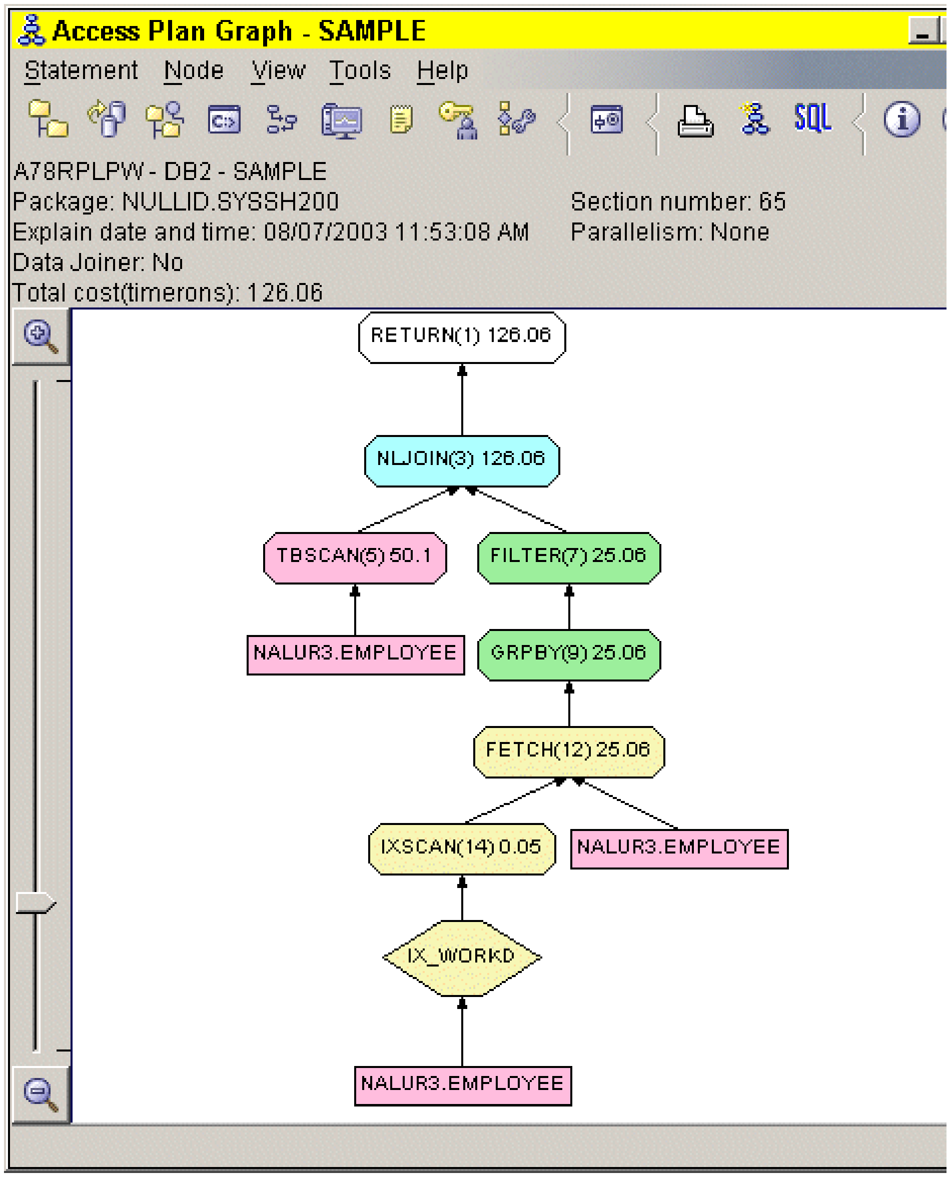
Query tag:

Optimization class: 5

[✓] Populate all columns in Explain tables

Figure 2-20 The Explain SQL Statement

Figure 2-21 on page 99 is the access path in graphical form and it has a bottom-up orientation; that is, it has to be interpreted from the bottom to the top.



A78RPLPW - DB2 - SAMPLE
Package: NULLID.SYSSH200
Section number: 65
Explain date and time: 08/07/2003 11:53:08 AM
Parallelism: None
Data Joiner: No
Total cost(timerons): 126.06

Figure 2-21 The Visual Explain access path

The geometric shapes displayed are called *nodes*; they represent the different access steps chosen by the optimizer. Tables are shown as rectangles, indexes are shown as diamonds, operators are shown as octagons, **TQUEUE** are shown as parallelograms, and functions are shown as hexagons.

For more information on the explain facility, refer to *DB2 UDB Administration Guide: Performance*, SC09-4821.

2.6.9 Heath Monitor and Health Center

DB2 UDB Version 8 has two new features to help monitor the health of the DB2 system, namely the Health Monitor and the Health Center. DB2 also provides a set of health indicators to evaluate specific aspects of database manager or database performance.

- **Health Monitor** is a server-side tool that periodically monitors the health of the DB2 instance, and raises an alert if a user-defined threshold has been reached or an abnormal state for an object is detected. When an alert is raised, an e-mail alert notification can be sent to the DBA or system administrator, and/or a script can be run from the Task Center.

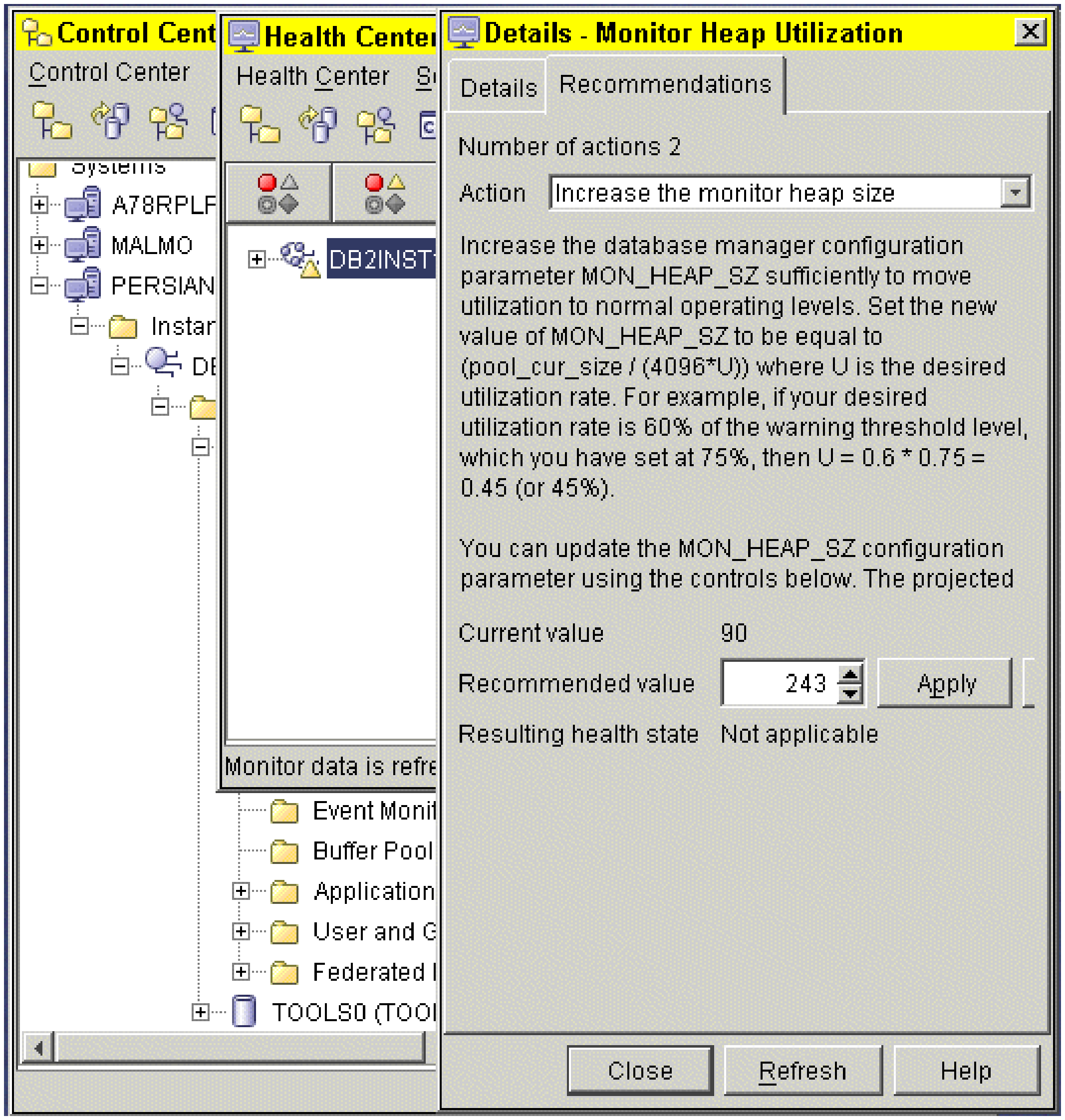
  Note: The Health Monitor uses a new interface to gather information about the health of the system, and imposes minimal performance overhead. It does not require the setting of any snapshot switches to collect information, and do not impose a performance penalty.

  The Health Monitor is enabled by default when an instance is created, and can disabled by setting the database manager configuration parameter **`HEALTH_MON`** to OFF.

  Health Monitor information can be accessed from Health Center, Web Health Center, the CLP, or via an API.

- **Health Center** provides the GUI interface to the Health Monitor; it can be used to configure the Health Monitor. The Health Center also provides access to information about a current alert, and the list of recommended actions that describe how to resolve the alert.

  You choose one of the recommended actions to address the alert. If the recommended action is to make a database or database manager configuration change, the new value will be recommended and may be applied immediately by clicking the **Apply** button, as shown in Figure 2-22 on page 101.



Number of actions 2

Action: Increase the monitor heap size

Increase the database manager configuration parameter MON_HEAP_SZ sufficiently to move utilization to normal operating levels. Set the new value of MON_HEAP_SZ to be equal to (pool_cur_size / (4096*U)) where U is the desired utilization rate. For example, if your desired utilization rate is 60% of the warning threshold level, which you have set at 75%, then U = 0.6 * 0.75 = 0.45 (or 45%).

You can update the MON_HEAP_SZ configuration parameter using the controls below. The projected

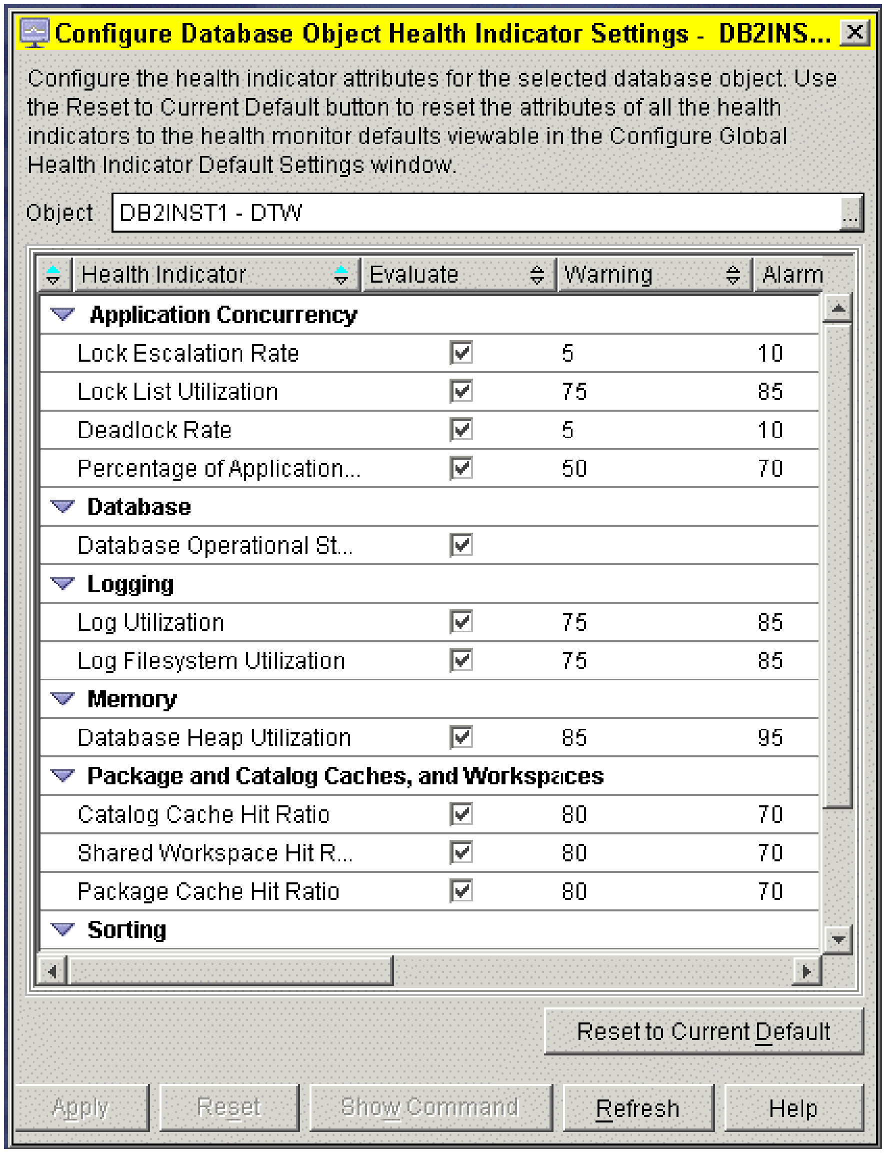
Current value 90

Recommended value 243

Resulting health state Not applicable

Figure 2-22 The Health Center

- **Health indicators** are used by the Health Center to evaluate specific aspects of database manager or database performance. A *health indicator* is a specific measurement tool to check the health of some aspect of database objects. It allows the administrator to specify the particular indicator to be evaluated, and the specific warning and alarm threshold to be applied, as shown in Figure 2-23 for a particular database (for example, **DTW**).


Configure Database Object Health Indicator Settings - DB2INS...

Configure the health indicator attributes for the selected database object. Use the Reset to Current Default button to reset the attributes of all the health indicators to the health monitor defaults viewable in the Configure Global Health Indicator Default Settings window.

Object: DB2INST1 - DTW

Health Indicator	Evaluate	Warning	Alarm
Application Concurrency			
Lock Escalation Rate	✓	5	10
Lock List Utilization	✓	75	85
Deadlock Rate	✓	5	10
Percentage of Application...	✓	50	70
Database			
Database Operational St...	✓		
Logging			
Log Utilization	✓	75	85
Log Filesystem Utilization	✓	75	85
Memory			
Database Heap Utilization	✓	85	95
Package and Catalog Caches, and Workspaces			
Catalog Cache Hit Ratio	✓	80	70
Shared Workspace Hit R...	✓	80	70
Package Cache Hit Ratio	✓	80	70
Sorting			

Figure 2-23 Health indicators

The command line interface also provides access and update capabilities to the health indicators.

The following commands list the health indicators for an instance, the description for a database level indicator **`db.spilled_sorts`**, the recommendations for a particular health indicator, and a new snapshot command to retrieve a state of an health indicator:

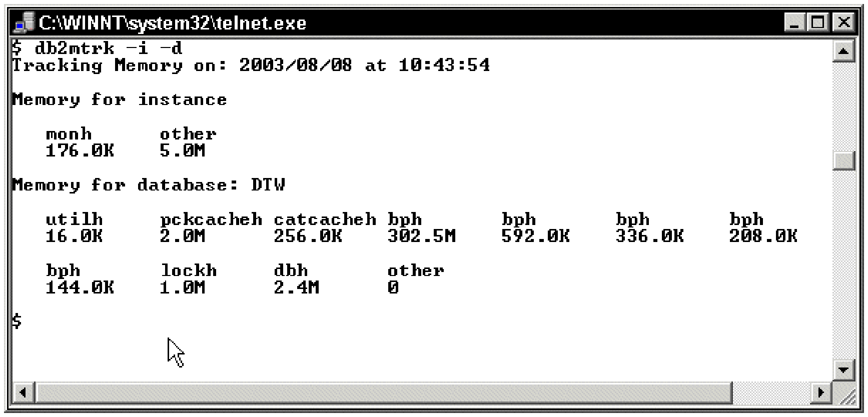
```
db2 get alert cfg for database manager

db2 get description for health indicator db.spilled_sorts

db2 get recommendations for health indicator db2.spilled_sort

db2 get health snapshot for database manager
```

For more details on health indicator attributes, refer to *DB2 UDB System Monitor Guide and Reference*, SC09-4847.

2.6.10 Memory Tracker

DB2 UDB Version 8 introduced the Memory Tracker tool db2mtrk, which provides a complete report of memory utilization for instances, databases and agents. The Memory Tracker will implicitly attach to the instance, without requiring the user to explicitly do so.

Figure 2-24 shows instance and database values.



```
$ db2mtrk -i -d
Tracking Memory on: 2003/08/08 at 10:43:54

Memory for instance

   monh      other
   176.0K    5.0M

Memory for database: DTW

   utilh     pckcacheh catcacheh bph       bph       bph       bph
   16.0K     2.0M      256.0K    302.5M    592.0K    336.0K    208.0K

   bph       lockh     dbh       other
   144.0K    1.0M      2.4M      0

$
```

Figure 2-24 Memory Tracker

For more details on Memory Tracker, refer to *DB2 UDB Command Reference*, SC09-4828.

2.6.11 Memory Visualizer

DB2 UDB Version 8 Memory Visualizer is new GUI interface for uncovering and fixing memory related problems of an instance. Memory Visualizer may be invoked from “Health Center recommendations”, or independently as its own monitoring tool from the Control Center by right-clicking the instance_name and selecting **View Memory Usage**.

The high-level memory components of Memory Visualizer are:

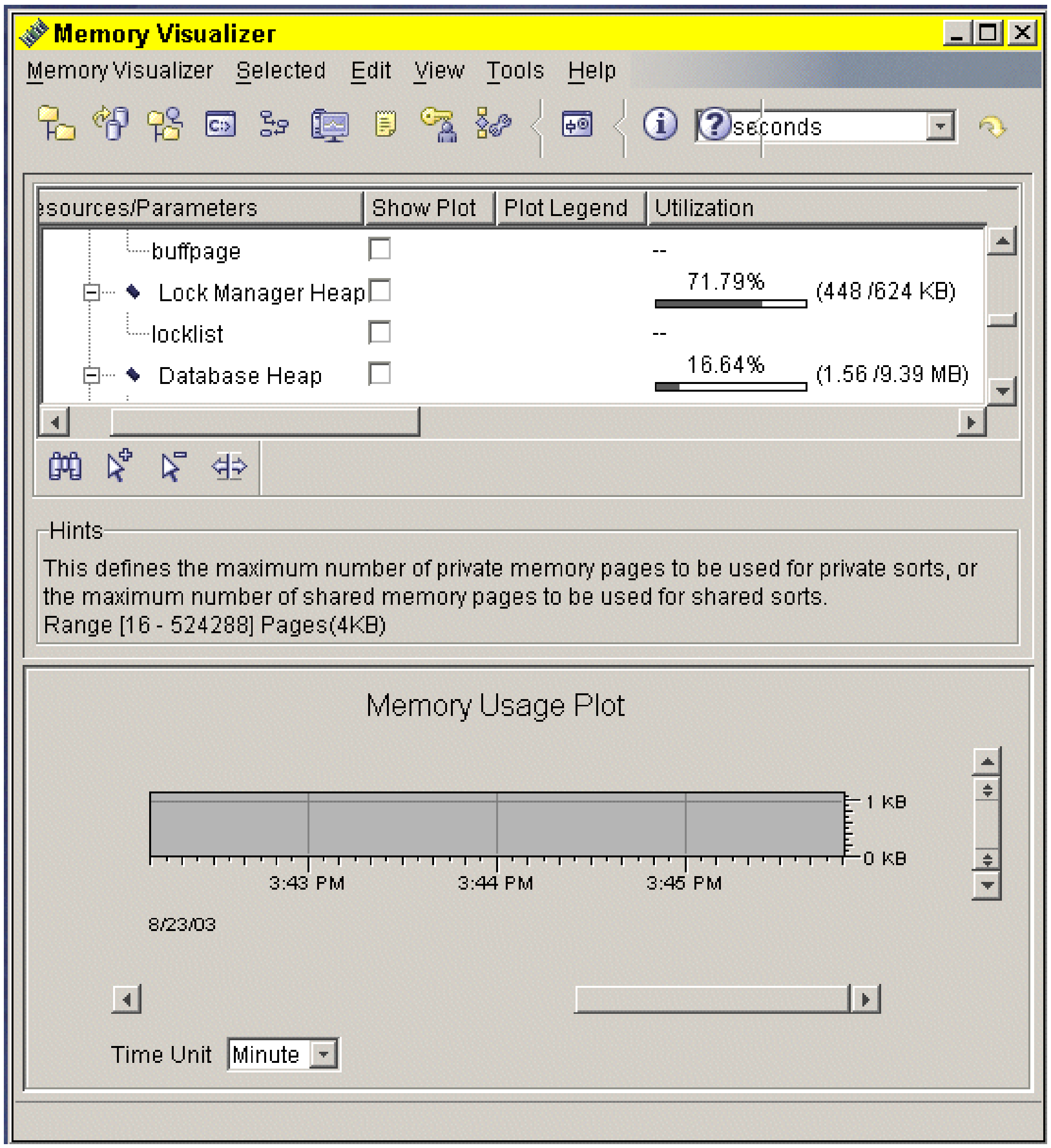
- Database manager shared memory
- Database global memory
- Application global memory
- Agent/Application shared memory
- Agent private memory

Each high-level memory component is divided into lower-level memory components that determine how the memory is allocated and de-allocated.

Memory Visualizer can perform the following tasks:

- View overall memory usage
- Update configuration parameters for an individual memory component to prevent it from using too much or too little memory
- Save memory allocation data
- Load memory allocation data from a file into a Memory Visualizer window

Figure 2-25 on page 105 provides an overview of Memory Visualizer. Historical values of alarm and warning thresholds are displayed for each memory component.



esources/Parameters	Show Plot	Plot Legend	Utilization
buffpage	☐		--
Lock Manager Heap	☐		71.79% (448 /624 KB)
locklist	☐		--
Database Heap	☐		16.64% (1.56 /9.39 MB)

Hints

This 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.
Range [16 - 524288] Pages(4KB)

Figure 2-25 The Memory Visualizer

3

Application design and system performance considerations

In this chapter we describe the key performance drivers that impact OLTP and BI performance, and suggest best practices for achieving superior DB2 OLTP and BI performance.

The topics covered are:

- OLTP and BI characteristics
- Key performance drivers
- Application design considerations
- System environment considerations

3.1 OLTP and BI characteristics

OLTP and BI environments have unique characteristics that require custom application and system tuning to achieve optimal performance. In the past, these two workload environments tended to be isolated from each other on different machines, thus making this tuning effort easier to manage. However, the differences between these workloads have begun to blur, with each workload accessing data in the other's domain. This trend may complicate the design and tuning of these mixed environments.

Attention: In this section, all performance recommendations assume that OLTP and BI environments are isolated from each other. We leave it to the reader to extrapolate the recommendations for these individual environments into their particular mixed workload environment.

However, it is reasonable to assume that tuning in a mixed workload environment needs to favor OLTP over BI because of OLTP's more stringent throughput and response time requirements.

The main characteristics of OLTP and BI environments are described here and highlighted in Table 3-1 on page 110.

3.1.1 OLTP characteristics

OLTP environments support day-to-day, mission-critical business activities such as order entry, stock trading, inventory management, and banking transactions. This typically involves hundreds to thousands of users issuing millions of transactions per day against large and small databases. Response time requirements tend to be subsecond and stringent, and actions need to be performed online and in real time.

OLTP workloads tend to have the following characteristics:

- Simple transactions, with each transaction issuing few simple SQL statements, accessing few rows, and performing few I/Os. No intra-query parallelism, few small sorts, and few indexes per table.
- Transactions perform a great deal of concurrent read and update activity.
- Predefined programmed applications.
- High throughput measured in hundreds of transactions per second, with the requirement for subsecond end-user response time.
- Hundreds to thousands of concurrent users.

- Volume of data may be very large (hundreds of gigabytes to a few terabytes) or just tens of gigabytes; however, each transaction typically accesses a few rows only.
- Potentially, long-running batch jobs (typically off-shift), with many “transactions” between commits.
- Very high availability requirements 7x24x365.

Performance of OLTP workloads are considerably enhanced by minimizing I/Os, optimizing CPU utilization, eliminating sorts, and improving concurrency between transactions.

3.1.2 BI characteristics

BI environments, on the other hand, involve gathering historical information from OLTP systems and external sources, producing reports, mining and analyzing information for trends and opportunities, and using the information thus gained to make timely and effective business decisions to gain a competitive edge in the marketplace. This typically involves a few users accessing very large databases, with low throughput requirements and response time in minutes or a few hours.

BI workloads tend to have the following characteristics:

- Medium-to-very complex queries, with each query issuing few complex SQL statements, accessing thousands to millions of rows, and performing hundreds to thousands of I/Os. Significant intra-query parallelism, large number and size of sorts, and many indexes per table.
- Queries are predominantly read only in nature.
- Canned, as well as ad hoc queries.
- Low throughput measured in tens to hundreds of queries per minute, with response time requirements of minutes to hours.
- Tens to hundreds of concurrent users.
- Volume of data may be very large (hundreds of gigabytes to many terabytes), with each query typically accessing thousands to millions of rows.
- Potentially, long-running extract and transformation jobs for loading data in to data warehouses.
- Potentially, long-running large reports and data mining activity.
- Moderate availability requirements, except in the case of real-time BI.

Performance of BI workloads are considerably enhanced by optimizing I/Os, having a large number of disks, promoting greater parallelism, eliminating sorts through indexing, and improving sort efficiency through adequate buffering and I/O placement.

Table 3-1 OLTP versus BI characteristics

Description	OLTP environments	BI environments
Mission-critical applications.	Yes.	Maybe.
Transaction profile.	Simple, with few SQL statements that typically return few rows and perform few I/Os. No intra-query parallelism, few small sorts, and few indexes per table.	Medium-to-complex SQL statements that typically return hundreds of rows and perform thousands of I/Os. Significant intra-query parallelism, large number and size of sorts, and many indexes per table.
Transaction profile.	Updates and reads, but mainly pre-programmed; no ad hoc.	Predominantly read only; mix of canned and ad hoc.
Throughput measure.	Hundreds to thousands of transactions per second.	Tens to hundreds of queries per minute.
Number of concurrent users.	Hundreds to thousands.	Tens to hundreds of users.
Volume of data involved.	Few gigabytes to hundreds of gigabytes to a few terabytes.	Hundreds of gigabytes to many petabytes.
Long running jobs?	Batch jobs (typically off-shift), with many transactions between commits.	Extract, transformation and load into data warehouses. Could also include large reports and data mining activity.
Availability requirements.	Very high 7x24x365.	Moderate to high.

3.2 Key performance drivers

The performance of an OLTP or BI environment depends upon many factors such as machine resources available, network bandwidth, and how well the application has been designed and the system environment tuned to address the specific performance objectives of the application workload.

In this chapter, we focus primarily on DB2-related performance drivers that impact *isolated* OLTP and BI environments (not mixed workloads) as follows:

- Identify the key performance drivers
- Discuss their performance considerations
- Suggest best practices for achieving superior performance with them
- Identify monitoring facilities to track their performance

We have categorized these key performance drivers as follows:

- Application design considerations
- System design considerations

3.3 Application design considerations

Any OLTP or BI application designed without performance in mind can have a major negative impact on overall system performance, which cannot be easily overcome with even the most optimal system tuning. For example, inefficient SQL and inappropriate locking strategies can result in suboptimal access path selection by the DB2 optimizer and significant locking contention, thereby increasing end-user response times and reducing throughput.

This subsection covers application performance considerations as they relate to:

- Table design
- Index design
- Table space design
- Writing efficient SQL
- Concurrency

3.3.1 Table design

DB2 supports four types of tables:

- “Regular” tables
- Materialized Query Tables (MQTs), formerly known as Automatic Summary Tables (ASTs)
- Multi-dimensional Clustering (MDC) tables
- Declared Global Temporary (DGTT) tables

Table design involves considering the following factors:

- Type of table
- Normalization
- Data types
- VALUE COMPRESSION and COMPRESS SYSTEM DEFAULT options
- Referential constraints

- Informational constraints
- MQT/AST design considerations
- MDC design considerations
- Declared Global Temporary Tables design considerations

We describe these considerations in the following subsections.

Refer to *DB2 UDB Administration Guide: Planning*, SC09-4822, for a more detailed discussion of table, index, and table space design considerations.

Type of table

While there are four types of tables, they are not all competing choices. The decisions to be made are as follows:

1. Should a “regular” table be used, or an MDC?
2. Should an MQT/AST be created or not?
3. Should a Declared Global Temporary Table (DGTT) or a “regular” table be used for intermediate results sets?

These choices are addressed here.

“Regular” table or MDC

The nature of the application workload determines whether or not an MDC is an appropriate choice for a table.

Certain kinds of applications, such as data warehousing involving “fact” tables and many “dimension” indexes, issue queries that retrieve data from the fact table along multiple “dimensions”.

- With “regular” tables, which support only a single clustering index, retrieval of large numbers of rows via a non-clustered index may be quite inefficient because of the need for multiple I/Os against the fact table. This is of particular concern with fact tables and multiple dimension indexes in data warehousing applications. In addition, a regular table’s degree of clustering tends to degrade over time as rows get added and deleted, necessitating a table reorganization to reestablish clustering.
- With MDCs, multiple clustering indexes corresponding to the dimensions may be defined on the fact table, and data is stored in a way that the degree of clustering does not degrade over time. This has significant performance benefits with data warehousing applications implementing fact tables and multiple dimensions. However, MDCs tend to consume additional disk storage to provide this benefit.

Performance considerations related to MDC design are covered in the IDUG May 2003 presentation, “New Data Clustering Techniques: Multidimensional Clustering” by Leslie Cranston.

MDC tables are therefore appropriate when an application requires data to be clustered along multiple keys for superior query performance, and the added costs of disk storage are acceptable.

MQT/AST to be created or not

Here again, the nature of the application workload determines whether or not an MQT/AST should be created.

In a data warehouse environment, users often issue queries repetitively against large volumes of data with minor variations in a query’s predicates. For example:

- Query A might request the number of items belonging to a consumer electronics product group sold in each month of the previous year for the western region.
- Query B may request the same kind of information for only the month of December for all regions in the USA.
- Query C might request monthly information for laptops for all regions in the USA over the past six months.

The results of these queries are almost always expressed as summaries or aggregates. The base data could easily involve millions of transactions stored in one or more tables, which would need to be scanned repeatedly to answer these queries. Query performance likely to be poor in such cases.

MQTs/ASTs provide a look-aside capability for such queries, which can result in orders of magnitude improvement in performance. When appropriate MQTs/ASTs are defined on base tables, queries that access the base tables are automatically rewritten (if appropriate) by the DB2 optimizer to access the MQTs/ASTs instead, in order to achieve superior query performance.

However, the cost of MQTs/ASTs are disk space, transaction overheads for refresh immediate MQTS/ASTs since they are updated synchronously, administration costs for loads, deferred refreshes, runstats, backups, and so on.

Attention: MQTs do *not* require an aggregate to be defined in order to be beneficial.

Performance considerations for MQTs/ASTs are discussed in “MQT/AST design considerations” on page 128, and the IBM Redbook *DB2 UDB’s High Function Business Intelligence in e-business*, SG24-6546.

MQTs/ASTs are therefore appropriate when a very large number of rows need to be scanned repeatedly to generate aggregates or summaries, and the added overheads of disk space and administration costs are acceptable.

DGTT or "regular" table for intermediate results sets

Certain kinds of applications create tables to process large amounts of data and drop those tables once the application has finished manipulating the data; DGTTs offer a viable alternative to "regular" tables for storing these intermediate results sets, for the following reasons:

- DGTTs are not recorded in the system catalog, are not persistent, are created in a user temporary table space, and are dropped at the end of the session. Therefore, they do not need to be managed like "regular" tables used for intermediate result sets, which need to be pruned and proliferation minimized when large numbers of such tables may be required.
- Unlike "regular" tables, there is no locking of DGTT rows, no catalog contention because there is no entry in the system catalog, and if the **`NOT LOGGED`** option is chosen, then neither the DGTT nor its contents are logged—all of which is probably acceptable for most intermediate result set processing. Setting the **`NOT LOGGED`** option not only has a positive impact on the performance of this application, but also on overall system performance, since it minimizes the amount of logging and contention for log buffers.

For further details about DGTTs, refer to *DB2 UDB Application Development Guide: Programming Client Applications*, SC09-4826.

DGTTs are therefore appropriate for applications that temporarily create tables for storing and manipulating large volumes of data, and then drop them when they are finished with it. Such applications tend to be more BI-oriented.

Note: Here again, it is very unlikely that OLTP applications perform operations that involve the creation, storing, and manipulation of intermediate results sets that are discarded at the end of the application. Therefore, DGTTs may not be appropriate for "typical" OLTP environments.

Normalization

The objective of normalization is to minimize redundancies in the data stored in different tables. Normalization improves the integrity of the data since SQL updates need to be applied to only a single table. The disadvantage of normalization is that query performance may deteriorate if a join is required to access data stored in different tables.

There are five normal forms, but a discussion of them is beyond the scope of this redbook. Refer to *An Introduction to Database Systems,* by Chris Date, ISBN 0-321-197844-4, for a detailed discussion of normalization.

At least Third Normal form is strongly recommended for OLTP applications since data integrity requirements are stringent, and joins involving large numbers of rows are minimal.

Data warehousing applications, on the other hand, are predominantly read only, and therefore benefit from denormalization, which involves duplicating data in one or more tables to minimize or eliminate joins. In such cases, adequate controls must be put in place to ensure that the duplicated data is always consistent in all tables to avoid data integrity issues.

Data types

DB2 supports a wide range of data types for the columns in a table. Data types can be categorized as:

- DB2-Supplied Data Types - these can be Numeric, String (Binary, Single Byte, Double Byte), or Date and Time.
- User Defined Data Types- these can be User Defined distinct Types (UDTs), User Defined Structured Types, or User Defined Reference Types.

> **Attention:** From a performance perspective, the use of User Defined Data Types should not impact response times. For example, User Defined distinct Types share the same code that is used by built-in data types to access built-in functions, indexes, and other database objects.

For more information on data types, refer to *DB2 UDB Administration Guide: Implementation*, SC09-4820.

Performance considerations

Data types should be defined to minimize disk storage consumption, enhance domain integrity[1], and avoid any unnecessary processing such as data type transformations.

Best practices

We recommend the following data types within the domain constraints required (what values need to be supported) by the application:

[1] DB2 automatically ensures a degree of integrity for values in a column based on the data type; for example, defining a data type as SMALLINT or INTEGER ensures that no alphabetic characters can be stored there, and defining a DATE data type ensures that date entered is always valid.

1. Use SMALLINT rather than INTEGER, and INTEGER rather than BIGINT where appropriate.
2. Use DATE or TIME rather than TIMESTAMP.
3. Use the DATE and TIME data types rather than CHAR.
4. Use NOT NULL for columns wherever possible.
5. Use CHAR rather than VARCHAR for any narrow columns (those that are less than 50 characters wide).
6. Choose VARCHAR instead of LONG VARCHAR when the row size is less than 32 K.
7. Use FLOAT rather than DECIMAL for columns if exact precision is not required.
8. Use IDENTITY columns, where appropriate.

VALUE COMPRESSION and COMPRESS SYSTEM DEFAULT

These options in the CREATE TABLE statement allow null and system default values to be compressed using a new row format.

- `COMPRESS SYSTEM DEFAULT` is specified at a column level, and directs DB2 to store default values with minimal space. Data types that support `COMPRESS SYSTEM DEFAULT` include all numerical type columns, fixed-length character, and fixed-length graphic string data types. This option can only be used if `VALUE COMPRESSION` is also specified.
- `VALUE COMPRESSION` is specified at the table level, and directs DB2 to store NULL and zero length data values for BLOB, CLOB, DBCLOB, LONG VARCHAR, and LONG VARGRAPHIC using minimal space in a new row format. Each null value is stored without using an additional byte.

  When this option is combined with the `COMPRESS SYSTEM DEFAULT` option, DB2 uses a new row format that allows system default values for the column to be stored more efficiently. Minimal disk space is used if the inserted or updated value is equal to the system default value for the data type of the column. The default value will not be stored on disk.

For further details on these new options, refer to *DB2 UDB Administration Guide: Implementation, SC09-4820* and *DB2 UDB SQL Reference Volume 2*, SC09-4845.

The trade-off is between saving disk space and increasing CPU consumption.

Note: Columns and rows remain compressed in the buffer pool, resulting in the ability to store more pages in the buffer pool. The performance of table scans are also improved with compression.

Consider the following recommendations:

- Use **`VALUE COMPRESSION`** if NULL and zero length values are common, or if column level compression (as specified by **`COMPRESS SYSTEM DEFAULT`**) is required.
- Use **`COMPRESS SYSTEM DEFAULT`** for columns that are mainly zeros or blank.

BI environments tend to require large volumes of data, and therefore benefit from data compression, particularly the performance of table scans, which occur more frequently in such environments. The CPU overhead tends to be a small percentage of the overall cost of a BI query.

OLTP environments should benefit from data compression as well, but the performance penalty of CPU is likely to be significantly higher per transaction that in the case of a BI query.

Referential constraints

Referential constraints are constraints placed on the data to enforce referential integrity (RI). RI is implemented by placing foreign keys in the child relationship of a parent-child relationship. Tables that are children of other tables can also be parents of other child tables.

While the primary function of referential constraints is to enforce RI, the DB2 optimizer is sometimes able to exploit the knowledge of RI between tables to optimize access paths of queries accessing parent and child tables.

For further details on referential constraints, refer to *DB2 UDB Administration Guide: Implementation*, SC09-4820.

OLTP environments should implement RI from a data integrity point of view, and indirectly benefit from the DB2 optimizer's ability to exploit RI relationships for query optimization.

BI environments tend to be predominantly read only and have their data loaded through a rigorous extract, transform, and load (ETL) process. They also tend to be significantly denormalized. Therefore, referential constraints may not be appropriate for such BI environments. Query optimization can still be achieved in such cases through the definition of informational constraints, as described in "Informational constraints".

Informational constraints

Informational constraints are essentially hints to the DB2 optimizer to help it determine the optimal access path for a query. Informational constraints are implemented as check constraints or referential constraints, and are defined in the **`CREATE/ALTER TABLE`** statement with the **`NOT ENFORCED`** option.

Informational constraints are *not* enforced by the database manager during updates to a table; they are used by the DB2 optimizer for potential query rewrite.

Note: The DB2 optimizer can be directed to ignore an informational constraint by specifying the **`DISABLE QUERY OPTIMIZATION`** option in the **`CREATE/ALTER TABLE`** statement.

Informational constraints can be used to improve the performance of queries with UNION ALL, as well as joins. It helps the optimizer rewrite outer joins as inner joins, and provides better estimates of cardinality.

Important: When using informational constraints, ensure that an appropriate process is in place to ensure the integrity of the data.

For further details on informational constraints, refer to *DB2 UDB Administration Guide: Implementation*, SC09-4820.

BI environments can benefit significantly from the definition of informational constraints since queries tend to be complex, volumes of data very large, and data integrity tends to be enforced through rigorous ETL processes.

OLTP environments are generally inappropriate for informational constraints since OLTP queries are very simple, and data integrity issues are paramount; referential constraints should be implemented in such environments. The exception could be for existing OLTP applications that have implemented user-defined referential integrity, and could benefit from informational constraints for query optimization.

3.3.2 MDC design considerations

An MDC is a table type that allows data to be independently clustered along more than one key[2], unlike “regular” tables, which can have their data clustered only according to a single key. Therefore, scans of an MDC table via any of the dimension indexes are equally efficient, unlike a regular table where only a scan of the data via the clustering index is likely to be efficient.

Additionally with MDC tables, clustering of data according to each key is guaranteed, therefore eliminating the need for a **`reorg`** to reestablish clustering.

[2] MDCs have wide applicability in “fact” tables in star schema implementations, and it is therefore quite common to see the word “dimensions” used instead of keys.

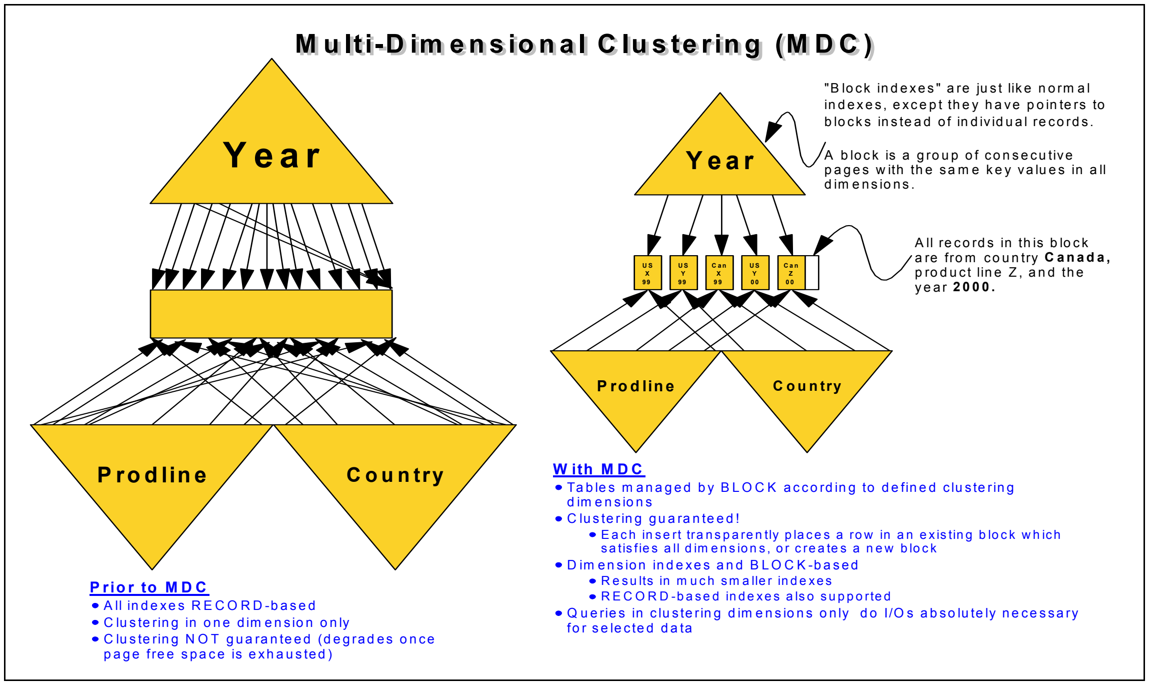
Note: This does *not* mean that a **`reorg`** is no longer necessary for MDCs. **`reorg`** may still be required against an MDC table to consolidate row overflows and unused space in extents due to deletions.

Unlike “regular” tables, an MDC table has multiple indexes associated with it when it is created. There is one block index per dimension, and one composite block index.

Also created is a block map (not shown in Figure 3-1) which is used to track the status of each block in the MDC table.

Note: A block is the smallest allocation unit of an MDC table. It is equivalent to an extent in “regular” tables.

Figure 3-1 highlights the differences between clustering in a regular table and multi-dimensional clustering in an MDC table.



"Block indexes" are just like normal indexes, except they have pointers to blocks instead of individual records.

A block is a group of consecutive pages with the same key values in all dimensions.

All records in this block are from country **Canada**, product line Z, and the year **2000**.

Prior to MDC
- All indexes RECORD-based
- Clustering in one dimension only
- Clustering NOT guaranteed (degrades once page free space is exhausted)

With MDC
- Tables managed by BLOCK according to defined clustering dimensions
- Clustering guaranteed!
    - Each insert transparently places a row in an existing block which satisfies all dimensions, or creates a new block
- Dimension indexes and BLOCK-based
    - Results in much smaller indexes
    - RECORD-based indexes also supported
- Queries in clustering dimensions only do I/Os absolutely necessary for selected data

Figure 3-1 Traditional RID clustering and Multidimensional clustering

DB2 manages MDC tables by block according to dimensions instead of RIDs as implemented in clustering indexes, as shown in Figure 3-2 on page 120.

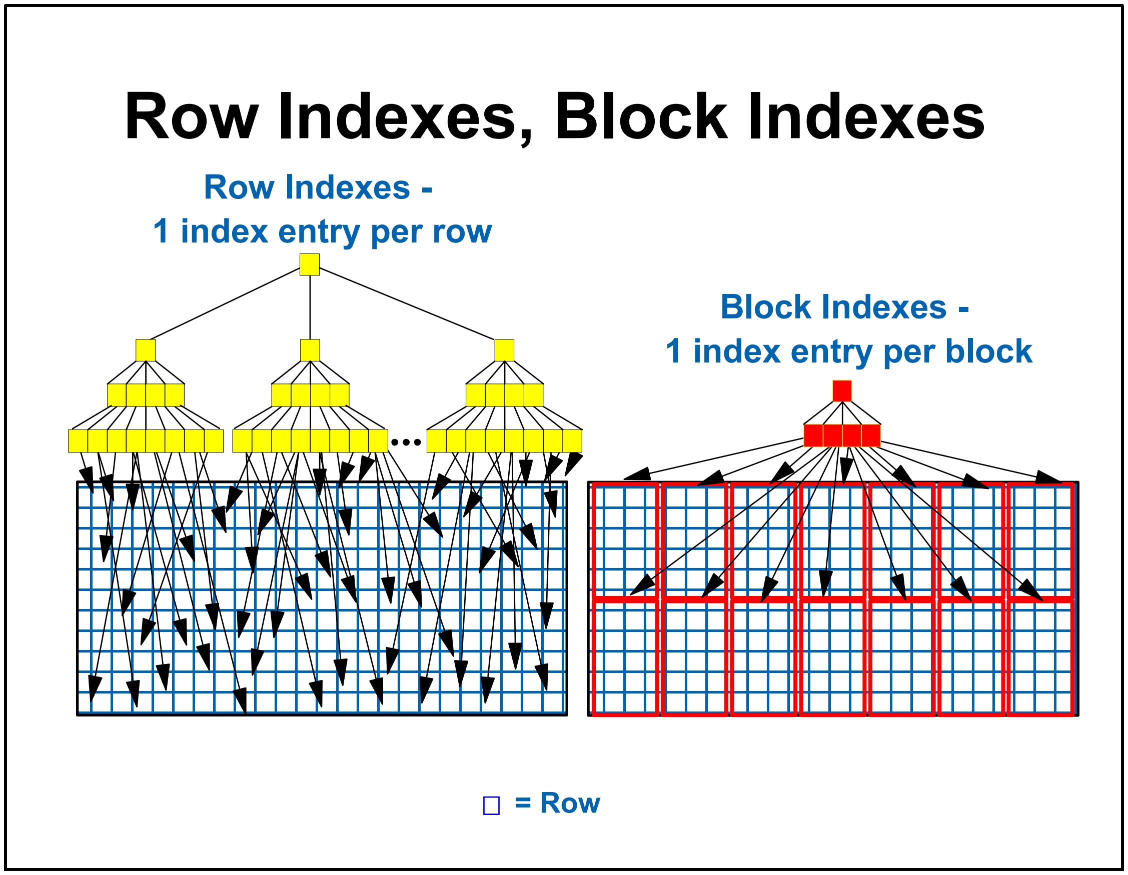


Figure 3-2   Row index vis-a-vis MDC block index

Figure 3-3 on page 121 presents a conceptual diagram of multi-dimensional clustering along three dimensions: region, year, and color.

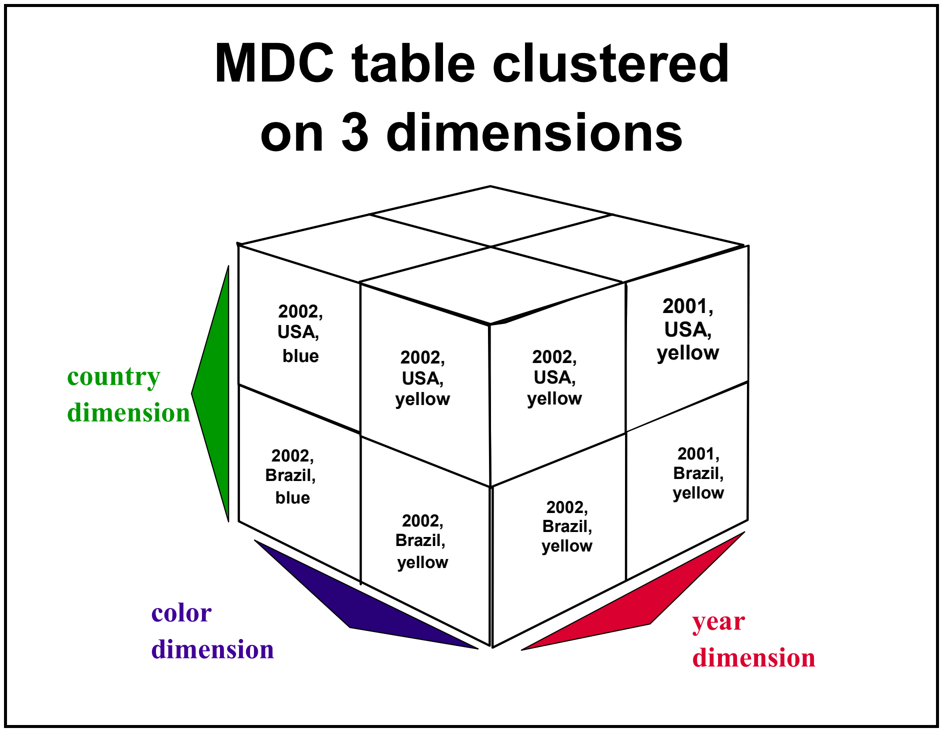


Figure 3-3 MDC dimensions

Each block contains only rows that have the same unique combination of dimension values. The set of blocks that have same unique combination of dimension values is called a *cell*. A cell may consist of one or more blocks in the MDC table, as shown in Figure 3-4 on page 122.

With MDC tables, clustering is guaranteed. If an existing block satisfies the unique combination of dimension values, the row is inserted into that block if there is sufficient space. If there is insufficient space in the existing block(s), or if no block exists with the unique combination of dimension values, a new block is created to store the row.

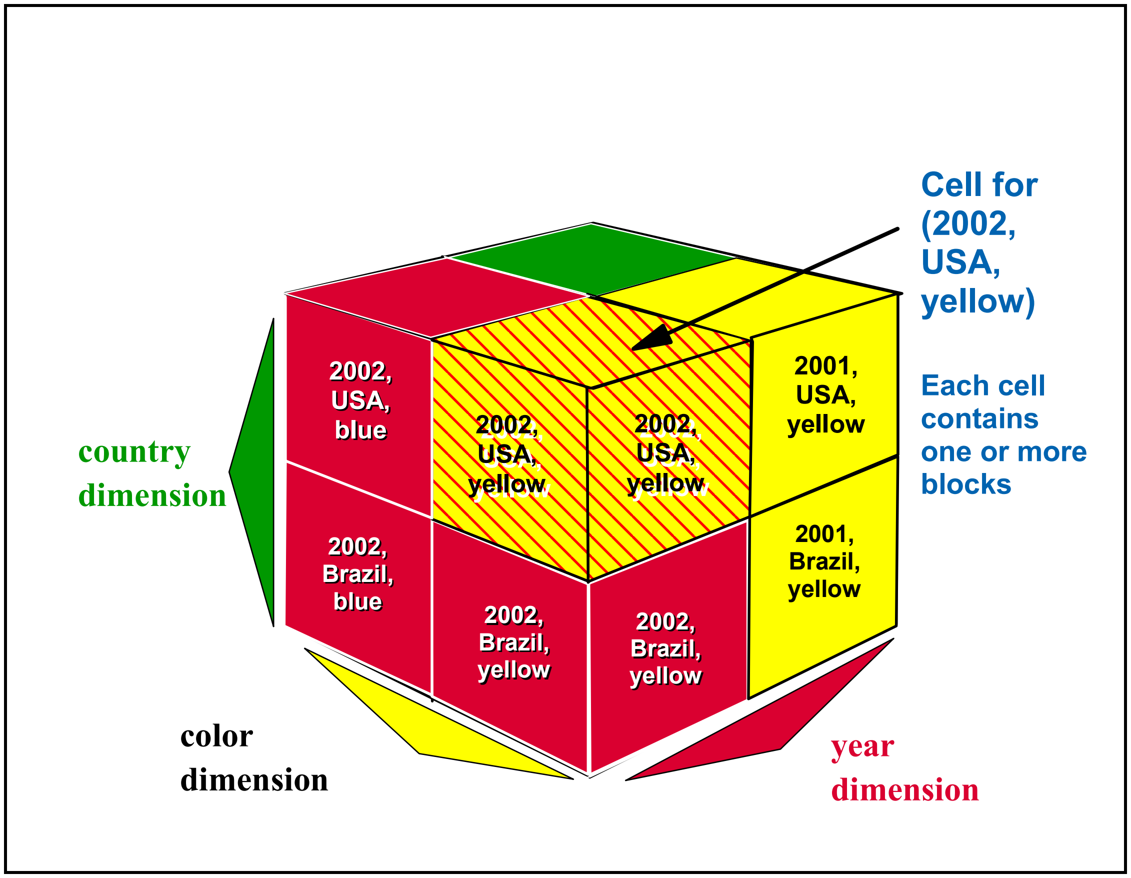


Figure 3-4 The cell for dimension values (2002, USA, yellow)

For further details on the structure of MDC tables and performance considerations, refer to the IDUG May 2003 presentation “New Data Clustering Techniques: Multidimensional Clustering” by Leslie Cranston.

Performance considerations

The performance of an MDC table is greatly dependent upon the proper choice of dimensions, and the block (extent) size of the table space for the given data and application workload.

A poor choice of dimensions and extent size can result in unacceptable disk storage utilization and poor query access performance, as well as `load` utility processing.

Choosing dimensions

The first step is to identify the queries in the in existing or planned workloads that can benefit from block-level clustering.

- For existing applications, the workload may be captured from the dynamic SQL snapshot and the SQL statement Event Monitor. The DB2 Query Patroller or other third party tools may also assist with such a determination.
- For future applications, this information will have to be obtained from requirements gathering.

Choosing the extent size

Extents are discussed in detail in “Extent size for the table space” on page 150. Extent size is related to the concept of *cell density*, which is the percentage of space occupied by rows in a cell. Since an extent only contains rows with the same unique combination of dimension values, significant disk space could be wasted if dimension cardinalities are very high; the worst case scenario is a dimension with unique values which would result in an extent per row.

The ideal MDC table is the one where every cell has just enough rows to exactly fill one extent. This can be difficult to achieve. The objective of this section is to outline a set of steps to get as close to the ideal MDC table as possible.

Note: The extent size is associated with a table space, and therefore applies to *all* of the dimension block indexes as well as the composite block index. This makes the goal of high cell density for every dimension block index and the composite block index very difficult to achieve.

Defining small extent sizes can increase cell density, but increase the number of extents per cell resulting in more I/O operations, and potentially poorer performance when retrieving rows from this cell. However, unless the number of extents per cell is excessive, performance should be acceptable. If *every* cell occupies more than one extent, then it can be considered to be excessive.

Sometimes, due to data skew, some cells will occupy a large number of extents while others will occupy a very small percentage of the extent. In such cases, it would signal a need for a better choice of dimension keys. Currently, the only way to determine the number of extents per cell requires the DBA to issue appropriate SQL queries or use **`db2dart`**.

Performance might be improved if the number of blocks could be reduced by consolidation. Unless the number of extents per cell is excessive, this situation is not considered a problem.

Note: The challenge is to find the right balance between sparse blocks/extents and minimizing the average number of extents per cell as the table grows to meet future requirements.

Best practices

We recommend the following best practices to achieve superior performance with MDCs:

1. Choose dimension column(s) that are good candidates for clustering as follows:
   - Columns used in high priority complex queries
   - Columns used in range, equality, and IN predicates such as:

     ```
     shipdate>’2002-05-14’, shipdate=’2002-05-14’, year(shipdate) in
     (1999, 2001, 2002)
     ```
   - Columns that define roll-in or roll-out of data such as:

     ```
     delete from table where year(shipdate) = ‘1999’
     ```
   - Columns with coarse granularity
   - Columns referenced in a GROUP BY clause
   - Columns referenced in an ORDER BY clause
   - Foreign key columns in “fact” table of a star schema database
   - Combinations of the above

   Note: Avoid columns that are updated frequently

2. If expressions are used to cluster data with generated columns, then the expression needs to be monotonic.

   Monotonic means that an increasing range of values on the base column corresponds to a range of values on the generated column that is never decreasing. For example:

   ```
   if (A > B) then expr(A) >= expr(B) and
   if (A < B) then expr(A) <= expr(B)
   ```

   In other words, as “A” increases in value, the expression based upon ”A” also increases or remains constant.

   Examples of monotonic operations include:

   ```
   A + B
   A * B
   integer(A).
   ```

Examples of non-monotonic operations are:

```
A - B
month(A)
day(A)
```

The expression `month(A)` is non-monotonic because as “A” increases, the value of the expression fluctuates as follows:

```
month(20010531) equals 05
month(20021031) equals 10
month(20020115) equals 01
```

So, as the date value increases, the value of the month fluctuates.

Note: If the SQL compiler cannot determine whether or not an expression is monotonic, the compiler assumes that the expression is *not* monotonic.

3. Do not choose too many dimensions without determining cell density; avoid too many sparse extents/blocks.
4. Once the dimensions have been selected, order them to satisfy the performance of high priority queries.

   When a MDC table is created, a composite block index is automatically created in addition to the dimension block indexes. While this index is used to insert records into the table, it can also be used like any other multi-column index as an access path for a query. Therefore, an appropriate ordering of the dimension columns can enhance the performance of certain types of queries with ORDER BY clauses and range predicates.
5. If disk space utilization of an MDC table is unacceptable or I/O performance is suffering because of too many extents in a cell, consider changing the following:
   – Extent size
   – Granularity of one or more dimensions
   – Number candidate dimensions
   – Different combination of dimensions

   Attention: Each of these changes requires the MDC table to be dropped and recreated. It is therefore vital to be very diligent during the design process to avoid having to make costly and time-consuming changes later.

MDCs are particularly suited for BI environments which involve star schemas, and queries that retrieve large numbers of rows along multiple dimensions. We strongly recommend that anyone considering a migration to an MDC table carefully model space utilization and cell utilization for candidate dimension keys, as well as the performance of high priority user queries, before committing to the selection of the dimension keys and extent size.

Monitoring performance metrics

Two elements to monitor with MDCs are as follows:

1. Space utilization
2. Efficacy of the extent size

Space utilization

One approach to determining whether the space utilization of an MDC table is acceptable is to compare the MDC storage requirement to that required for a non-MDC table.

For the existing MDC table, we recommend the following:

1. The table should be reorganized or freshly loaded.
2. Execute **`runstats`** to update table statistics.
3. Review the **`FPAGES`** (total number of pages in **`SYSCAT.TABLES`**) utilization statistic for the table to accurately determine the current number of pages allocated and to provide a baseline for evaluating dimension choices.

The equivalent space consumption for a non-MDC table requires using the formulae described in *DB2 UDB Administration Guide: Planning*, SC09-4822.

A significant (about 25%) difference in space utilization between the MDC table and an equivalent non-MDC table points to a poor selection of dimension keys, which results in unused space due to sparse extents/blocks. This may require changing the dimension keys as well as the extent size, which requires an existing MDC table to be dropped and recreated.

Note: Accommodating for future growth in the size of an MDC table may manifest transitory sparse extents/blocks, and should be evaluated carefully before initiating changes.

Efficacy of the extent size

The following steps can help determine the efficacy of an MDC table’s extent size:

1. Determine the number of cells
2. Determine the space occupied per cell
3. Determine cell utilization

Determine the number of cells

Example 3-1 provides the exact number of cells for an existing non-MDC table and a candidate dimension set. The number of cells will be equal to the number of unique combinations of the dimension attributes.

Example 3-1 Counting the number of cells

```
WITH cell_table as (
   SELECT DISTINCT dimension_col1, dimension_col2, ... , dimension_colN
   FROM table )
SELECT COUNT(*) as cell_count
   FROM cell_table
```

For a planned MDC table, the upper bound for the number of cells can be determined by computing the Cartesian product of the expected number of unique values in each dimension column.

Note: This estimate will be inaccurate if there are correlations between the dimension columns.

Determine the space occupied per cell

To determine the space needed to store each cell, we need to determine the number of rows in the table (**`SELECT COUNT(*) FROM table_name`** for an existing table; otherwise, use your best guess), and the extent size in bytes (**`SELECT (PAGESIZE * EXTENTSIZE) FROM SYSCAT.TABLESPACES WHERE TBSPACE LIKE ‘tablespace_name%’`**).

Example 3-2 provides the SQL to compute the average (**`RpC`**), minimum (**`MinRpC`**) and maximum (**`MaxRpC`**) rows per cell for the combination of dimensions of an existing MDC table.

Example 3-2 SQL to examine rows per cell

```
WITH cell_table (dimension_col1, dimension_col2, ... , dimension_colN, RpC)
   as (SELECT DISTINCT dimension_col1, dimension_col2, ... , dimension_colN,
         COUNT(*) FROM table
      GROUP BY dimension_col1, dimension_col2, ... , dimension_colN)
SELECT avg(RpC) as RpC, min(RpC) as MinRpC, max(RpC) as maxRpC
   FROM cell_table
```

The space occupied by rows in a cell (**`SpC`**) can be computed as follows:

```
   SpC = RpC x (average row size)
```

Determine cell utilization

The space utilization of a cell can now be computed as follows:

```
Cell utilization = SpC / (extent size in bytes)
```

This ratio provides a measure of the efficacy of the extent size for the current set of dimensions.

- A cell utilization value of 1.0 indicates that one cell will fully occupy one extent.

  This is the optimal value, and the candidate dimension set appears appropriate, assuming end-user queries will benefit from these dimensions. Compare the space required by the MDC table against the baseline from the non-MDC table.

- A really large utilization value (say, 10 or higher) indicates that multiple extents are required to store the rows in a cell. Performance of I/O operations may be improved by increasing the size of the extent, since this would result in fewer I/O operations.

  As cell utilization gets higher, performance of queries may be improved by adding additional dimension keys.

- An extremely small utilization (0.1 or less) indicates that a great deal of storage space allocated to the table will be unused.

  An MDC table with such cell utilization could require ten times the space consumed by a non-MDC table. This may require changing the dimension keys as well as the extent size, which requires an existing MDC table to be dropped and recreated.

3.3.3 MQT/AST design considerations

An MQT/AST is a table whose structure and contents are based on an SQL query. The SQL query used in defining the MQT/AST may access one or more tables.

MQTs were designed to improve the performance of queries in a data warehousing environment where users often issue queries repetitively against large volumes of data with minor variations in a query's predicates, as discussed in "MQT/AST to be created or not" on page 113. As mentioned earlier, MQTs do not require an aggregate function in its definition to be beneficial.

MQTs/ASTs provide a look-aside capability for such queries that can result in orders of magnitude improvement in performance, as shown in Figure 3-5 on page 129. When appropriate, MQTs/ASTs are defined on base tables, and queries that access the base tables are automatically rewritten (if appropriate) by the DB2 optimizer to access the MQTs/ASTs instead, in order to achieve superior query performance.

Note: Prior to DB2 UDB V8, the term Automatic Summary Tables (ASTs) was used for these look-aside tables in IBM product documentation. While it was possible to define non-aggregate ASTs in DB2 UDB V7, the restriction was that such an AST could only be defined on single base table.

In DB2 UDB V8, this restriction was removed, and since these look-aside tables can include other than summary data, the more generalized term Materialized Query Tables (MQT) was introduced. ASTs can be considered to be a subset of the generalized MQT which specifically includes summary data.

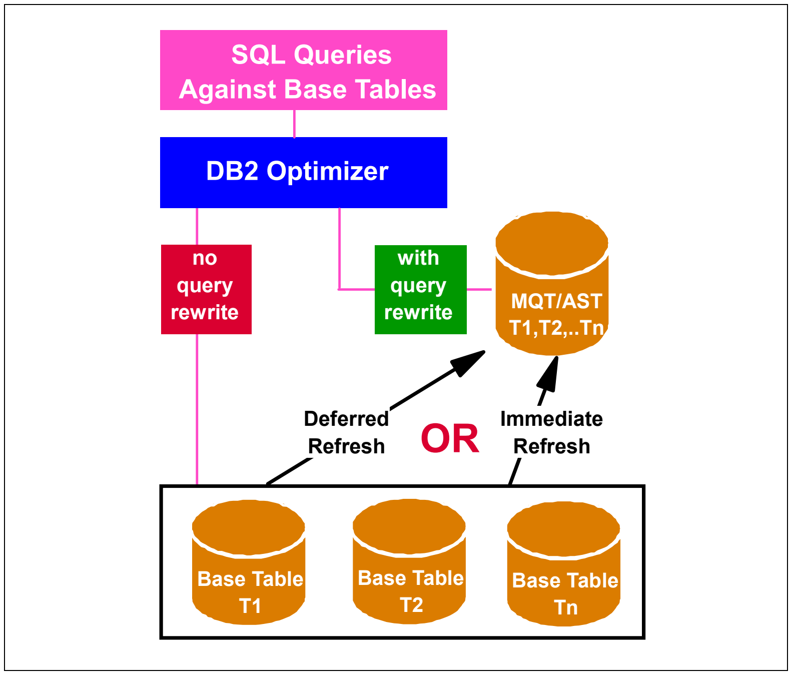


Figure 3-5 MQT/AST look-aside concept

MQTs/ASTs exploitation involves the following:

1. Have a DBA pre-compute an aggregate query, and materialize the results into a table. This summary table would contain a superset of the information that could answer a number of queries that had minor variations.

2. Take advantage of the DB2 optimizer‘s automatically rewrite capability to target the MQTs/ASTs instead (if appropriate) in order to satisfy the original query.

Since the MQT/AST often contains precomputed summaries and/or a filtered subset of the data, it would tend to be much smaller in size than the base tables from which it was derived. When a user query accessing the base table is automatically rewritten by the DB2 optimizer to access the materialized view instead, then significant performance gains can be achieved.

Important: MQTs/AST’s functionality is somewhat similar to the role of a DB2 index ,which provides an efficient access path that the query user is typically unaware of. However, unlike an index, a user may directly query the MQT/AST, but this is not generally recommended since it would detract from the appeal of an MQT/AST being a black box that an administrator creates and destroys as required to deliver superior query performance.

Adapting their queries to use an MQT/AST may not be a trivial exercise for the user.

There are two approaches to refresh MQTs/ASTs, deferred or immediate, as shown in Figure 3-5 on page 129:

1. In the deferred refresh approach, the contents of the MQT/AST are not kept in sync automatically with the base tables when they are updated. In such cases, there may be a latency between the contents of the MQT/AST and the contents in the base tables.

   Figure 3-6 on page 131 provides an overview of the deferred refresh mechanism.

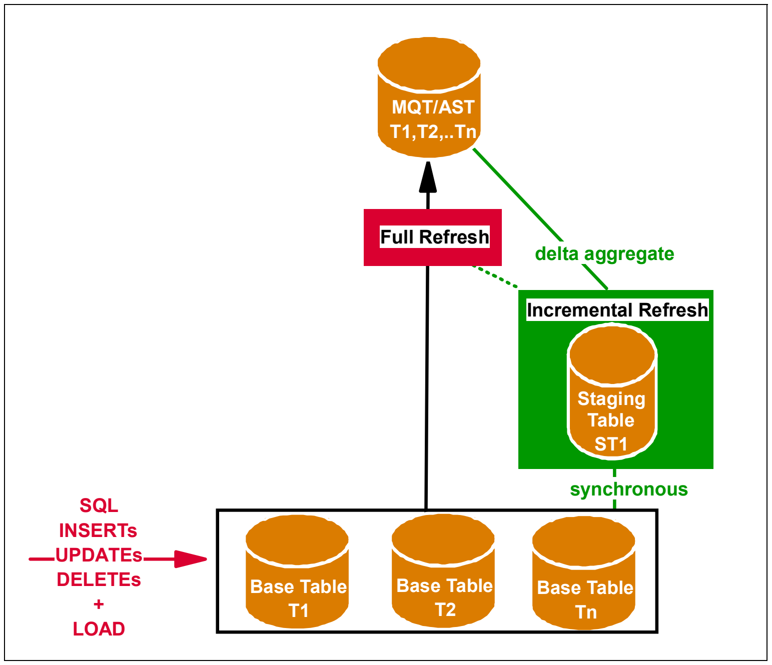


Figure 3-6 Deferred Refresh mechanism

`REFRESH DEFERRED` tables can be updated via the `REFRESH` table command with either a full refresh (`NOT INCREMENTAL`) option, or an incremental (`INCREMENTAL`) option.

For further details, refer to *DB2 UDB's High Function Business Intelligence in e-business*, SG24-6546.

2. In the immediate refresh approach, the contents of the MQT/AST are always kept in sync with the base tables. An update to an underlying base table is immediately reflected in the MQT/AST as part of the update processing.

   Other users can see these changes after the unit of work has completed on a commit. There is no latency between the contents of the MQT/AST and the contents in the base tables.

Table 3-2 summarizes some of the considerations relating to refresh immediate and refresh deferred MQTs/ASTs.

Table 3-2 Refresh considerations

Items	REFRESH IMMEDIATE	REFRESH DEFERRED	
	System maintained only	**System maintained**	**User maintained only**
Static SQL	Optimization	No optimization	No optimization
Dynamic SQL	Optimization	Optimization	Optimization
SQL INSERT, UPDATE, DELETE against materialized view	Not permitted	Not permitted	Permitted
REFRESH TABLE tablename	Permitted	Permitted	Not applicable
REFRESH TABLE NOT INCREMENTAL	Permitted	Permitted	Not applicable
REFRESH TABLE INCREMENTAL	Permitted	Requires staging table	Not applicable
Staging table	Not applicable	Same restrictions to creating them, as those applying to REFRESH IMMEDIATE materialized views	Not applicable

DB2 supports MQTs/ASTs that are either maintained by the system, which is the default, or maintained by the user, as follows:

- **`MAINTAINED BY SYSTEM`** (default)

  In this case, DB2 ensures that the MQTs/ASTs are updated when the base tables on which they are created get updated. Such MQTs/ASTs may be defined as either **`REFRESH IMMEDIATE`**, or **`REFRESH DEFERRED`**. If the **`REFRESH DEFERRED`** option is chosen, then either the **`INCREMENTAL`** or **`NON INCREMENTAL`** refresh option can be chosen during refresh.

- **`MAINTAINED BY USER`**

  In this case, it is up to the user to maintain the MQTs/ASTs whenever changes occur to the base tables. Such MQTs/ASTs *must be* defined with the **`REFRESH DEFERRED`** option. Even though the **`REFRESH DEFERRED`** option is required, unlike **`MAINTAINED BY SYSTEM`**, the **`INCREMENTAL`** or **`NON INCREMENTAL`** option does not apply to such MQTs/ASTs, since DB2 does not maintain such MQTs/ASTs. Tw0 possible scenarios where such MQTs/ASTs could be defined are as follows:

a. For efficiency reasons, when users are convinced that they can implement MQTs/ASTs maintenance far more efficiently than the mechanisms used by DB2 (for example, the user has high performance tools for rapid extraction of data from base tables, and loading the extracted data into the MQTs/ASTs).

b. For leveraging existing "user maintained" summaries, where the user wants DB2 to automatically consider them for optimization for new ad hoc queries being executed against the base tables.

Performance considerations

MQTs/ASTs have the potential to provide significant performance enhancements to certain types of queries, and should be a key tuning option in every DBA's arsenal. Like any other table, defining appropriate indexes on MQTs/ASTs and ensuring that their statistics are current will increase the likelihood of their being used by the DB2 optimizer during query rewrite, and enhance the performance of queries that use them.

However, MQTs/ASTs have certain overheads which should be carefully considered when designing them. These include:

- Disk space, due to the MQTs/ASTs and associated indexes, as well as staging tables.
- Locking contention on the MQTs/ASTs during a refresh.
- With deferred refresh, the MQT/AST is offline while the **`REFRESH TABLE`** is executing.
- The same applies to the staging table if one exists. Update activity against base tables may be impacted during the refresh window.
- With immediate refresh, there is contention on the MQTs/ASTs when aggregation is involved due to SQL insert, update, and delete activity on the base table by multiple transactions.
- Logging overhead during refresh of very large tables.
- Logging associated with staging tables.
- Response time overhead on SQL updating the base tables when immediate refresh and staging tables are involved, because of the synchronous nature of this operation.

Best practices

We recommend the following best practices to achieve superior performance with MQTs/ASTs:

1. The main objective should be to minimize the number of MQTs/ASTs required by defining sufficiently granular **`REFRESH IMMEDIATE`** and **`REFRESH DEFERRED`** MQTs/ASTs that deliver the desired performance, while minimizing their overheads.

   Figure 3-7 on page 135 provides an overview of the steps involved in designing **`REFRESH DEFERRED`** MQTs/ASTs.

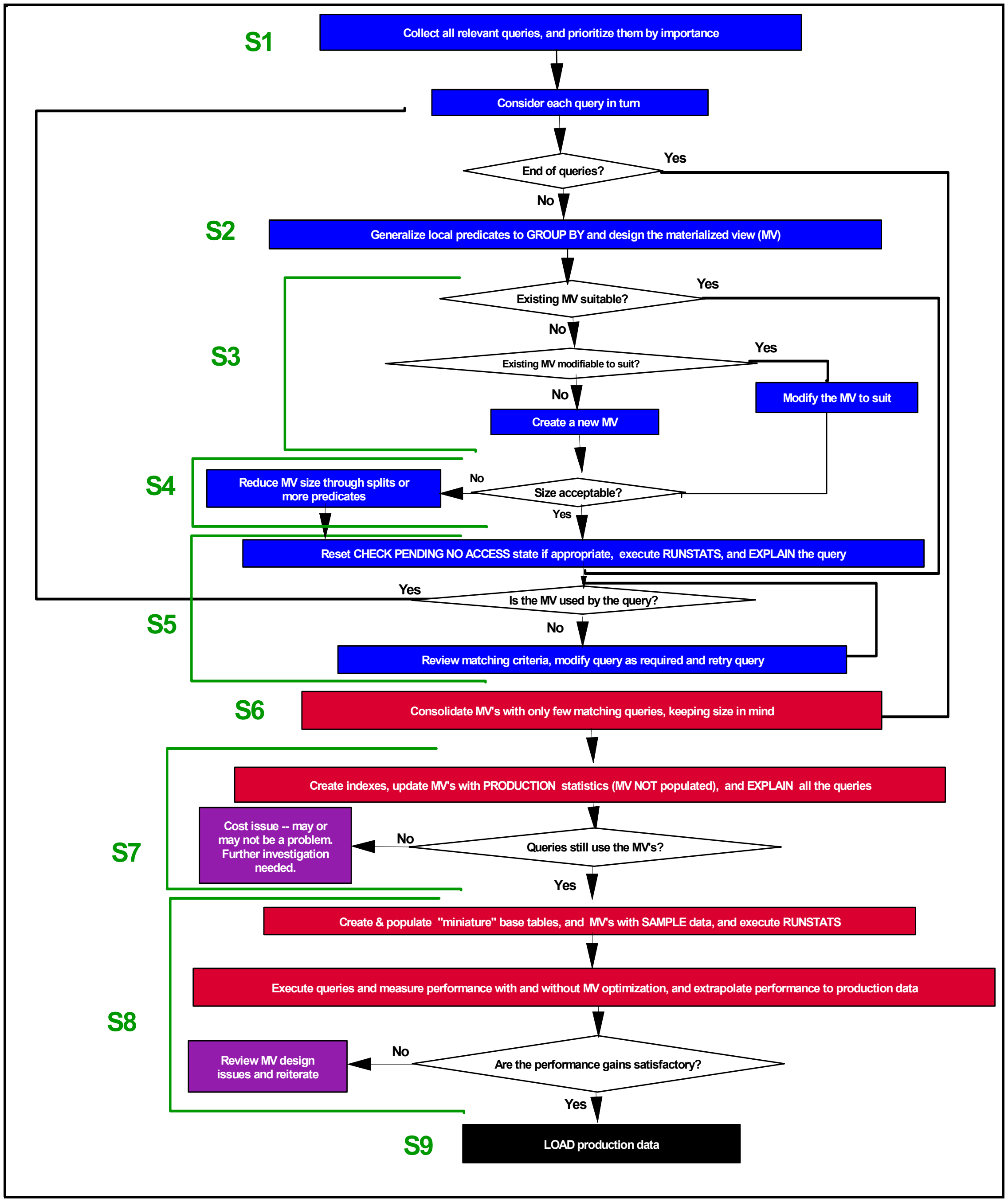

S1
Collect all relevant queries, and prioritize them by importance
Consider each query in turn
End of queries?
Yes
No
S2
Generalize local predicates to GROUP BY and design the materialized view (MV)
Existing MV suitable?
Yes
No
S3
Existing MV modifiable to suit?
Yes
Modify the MV to suit
No
Create a new MV
S4
Reduce MV size through splits or more predicates
No
Size acceptable?
Yes
Reset CHECK PENDING NO ACCESS state if appropriate, execute RUNSTATS, and EXPLAIN the query
Yes
Is the MV used by the query?
S5
No
Review matching criteria, modify query as required and retry query
S6
Consolidate MV's with only few matching queries, keeping size in mind
Create indexes, update MV's with PRODUCTION statistics (MV NOT populated), and EXPLAIN all the queries
Cost issue -- may or may not be a problem. Further investigation needed.
No
S7
Queries still use the MV's?
Yes
Create & populate "miniature" base tables, and MV's with SAMPLE data, and execute RUNSTATS
Execute queries and measure performance with and without MV optimization, and extrapolate performance to production data
S8
Review MV design issues and reiterate
No
Are the performance gains satisfactory?
Yes
S9
LOAD production data

Figure 3-7 Overview of the design of REFRESH DEFERRED MQTs/ASTs

Note: DB2 plans to provide an MQT Design Advisor wizard in the future to assist DBAs in defining effective MQTs/ASTs for their environments.

2. When an MQT/AST has many tables and columns in it, it is sometimes referred to as a "wide" MQT/AST. Such an MQT/AST allows a larger portion of a user query to be matched, and hence provides better performance. However, when the query has fewer tables in it than in the MQT/AST, we need to have declarative or informational referential integrity constraints defined between certain tables in order for DB2 to use the MQT/AST for the query as discussed in "Informational constraints" on page 117. Note that a potential disadvantage of "wide" MQT/AST is that they not only tend to consume more disk space, but may also not be chosen for optimization because of the increased costs of accessing them.
3. When an MQT/AST has fewer columns and/or tables, it is sometimes referred to as a "thin" MQT/AST. In such cases, we reduce space consumption at the cost of performing joins during the execution of the query. For example, we may want to only store aggregate information from a fact table (in a star schema) in the MQT/AST, and pick up dimension information from the dimension tables through a join. Note that in order for DB2 to use such a MQT/AST, the join columns to the dimension tables must be defined in the MQT/AST. Note also that referential integrity constraints requirements do not apply to "thin" MQTs/ASTs.
4. Incremental refresh should be used to reduce the duration of the refresh process. For the duration of a full refresh, DB2 takes a share lock on the base tables, and a z-lock on the MQT/AST. Depending upon the size of the base tables, this process can take a long time. The base tables are not updateable for this duration, and the MQT/AST is not be available for access or optimization either. Incremental refresh can reduce the duration of the refresh process, and increase the availability of the base tables and the materialized view. Incremental refresh should be considered when one or more of the following conditions exist:
   - The volume of updates to the base tables relative to size of the base tables is small.
   - The duration of read only access to the base tables during a full refresh is unacceptable.
   - The duration of unavailability of the MQT/AST during a full refresh is unacceptable.

For further details on all these recommendations, refer to *DB2 UDB's High Function Business Intelligence in e-business*, SG24-6546.

MQTs/ASTs are particularly suited for BI environments that involve queries that retrieve large numbers of rows and compute aggregations/summaries of these rows.

Monitoring performance metrics

A useful MQT/AST is one that is used in a query re-write by the DB2 optimizer and delivers significant performance benefits.

DB2 Explain identifies whether a particular query is rewritten by the DB2 optimizer to exploit and MQT/AST. Whether or not this actually results in an improvement in query performance can only be ascertained by actually measuring the query's performance with and without MQT/AST access.

Note: The DB2 optimizer may choose to ignore the MQT/AST for a query rewrite of a user query for a number of reasons, including:

- State of the MQT/AST, such as CHECK PENDING
- QUERY OPTIMIZATION LEVEL
- CURRENT REFRESH AGE special register value
- Matching criteria for query rewrite being inhibited
- Costs of using the MQT/AST to satisfy the query being suboptimal

There is no direct mechanism to determine the specific reason (suboptimal cost, or matching inhibited, or other) why the DB2 optimizer chose not to consider an MQT/AST for query rewrite.

3.3.4 Index design

Indexes provide the following functionality:

- Enforcement of the uniqueness constraints on one or more columns
- Efficient access to data in underlying tables when only a subset of the data is required, or when it is faster than scanning the entire table.

Indexes can therefore be used to:

- Ensure uniqueness
- Eliminate sorts
- Avoid table scans where possible
- Provide ordering
- Facilitate data clustering for more efficient access
- Speed up table joins

DB2 provides the Design Advisor wizard to recommend indexes for a specific query or workload. It can assist the DBA in determining indexes on a table that are not being used.

DB2 UDB Version 8 introduced a new type of index called a Type 2 index, which offers significant concurrency and availability advantages over the previous index structure (Type 1 index). For details on the structure and concurrency characteristics of Type 2 indexes, refer to *DB2 UDB Administration Guide: Performance*, SC09-4821.

Attention: Type 1 and Type 2 indexes cannot coexist on the same table. All indexes on a table must be of the same type.

Performance considerations

While indexes have the potential to significantly reduce a query's access time, the trade-off is in disk space utilization, slower updates (SQL **`INSERT, UPDATE`**, and **`DELETE`**s), locking contention, and administration costs (**`runstats`**, **`reorg`**). Each additional index potentially adds an alternative access path for a query for the optimizer to consider, which increases the compilation time.

Note: Type 2 indexes consume more space than Type 1 indexes.

Best practices

We recommend the following best practices to achieve superior index performance:

1. Use the Design Advisor, as described in 2.6.7, "Design Advisor" on page 93, to find the best indexes for a specific query or for the set of queries that defines a workload.
2. To eliminate some sorts and define primary keys and unique keys, wherever possible.
3. Add INCLUDE columns to unique indexes to improve data retrieval performance. Good candidates are columns that:
   - Are accessed frequently and would therefore 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
   - Are updated infrequently
4. To access small tables efficiently, use indexes to optimize frequent queries to tables with more than a few data pages. Create indexes on the following:

   - Any column you will use when joining tables
   - Any column from which you will be searching for particular values on a regular basis
5. 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 by specifying the **`ALLOW REVERSE SCANS`** parameter in the **`CREATE INDEX`** statement, scans in the specified index order perform slightly better than reverse scans.
6. 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. Unnecessary indexes also impact update performance in OLTP environments.
7. To improve performance of delete and update operations on the parent table, create indexes on foreign keys.
8. For fast sort operations, create indexes on columns that are frequently used to sort the data.
9. 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.
10. To help keep newly inserted rows clustered according to an index, define a clustering index. Clustering can significantly improve the performance of operations such as prefetch and range scans. Only one clustering index is allowed per table. A clustering index should also significantly reduce the need for reorganizing the table.

    Use the **`PCTFREE`** keyword when you define the index 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.
11. 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 before an online index defragmentation is attempted. 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.

12. The **PCTFREE** parameter in the **CREATE INDEX** statement specifies the percentage of each index leaf page to leave as free space. For non-leaf pages, it will choose the value you specify—unless the value that you specify is less than 10%, in which case the 10% value is chosen.

    Choose a smaller value for **PCTFREE** to save space and index I/Os in the following cases:

    – Index is never updated.

    – Index entries are in ascending order and mostly high-key values are inserted into the index.

    – The index entries are in descending order and mostly low-key values are inserted into the index.

    A larger value for **PCTFREE** should be chosen if the index gets updated frequently in order to avoid page splits, which reduce performance because they result in index pages no longer being sequential or contiguous. This has a negative impact on prefetching, and potentially space consumption as well, depending upon the key values being inserted/updated.

13. Ensure that the number of index levels in the index tree are kept to a minimum (less than 4, if possible); this is the **NLEVELS** column in the **SYSCAT.INDEXES** catalog table. The number of levels in the index is affected by the number of columns in the key and the page size of the table space in which it is created.

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:

1. For OLTP environments with transactions that perform updates, have as few indexes as possible without compromising performance (typically one or two indexes per table).
2. Because tables in BI environments are very large, and the queries are primarily read-only, multiple indexes (typically more than 5) should be defined for optimal query performance.
3. For OLTP environments, you might create between two and five indexes.

For further details, refer to *DB2 UDB Administration Guide: Performance*, SC09-4821.

Performance monitoring metrics

The efficacy of an index is ultimately measured by whether or not it is used in queries whose performance it is meant to improve.

For dynamic SQL, DB2 Explain is the mechanism for determining whether or not indexes are being used in queries.

For static SQL statements, package dependencies are recorded in the catalog table **SYSCAT.PACKAGEDEP**, which contains a row for each dependency that packages have on indexes, tables, views, triggers, functions, aliases, types, and hierarchies.

3.3.5 Table space design

A *table space* is a database object that is used by DB2 to specify the physical location of data in a database; it is the layer between the table metadata and the actual container that holds table data. Tables and indexes reside within tablespaces. A table space can contain one or many tables/indexes.

DB2 supports the following types of table spaces:

- **REGULAR TABLE SPACE** is used to store all data except for temporary tables.

  REGULAR is the default table space type.
- **LARGE TABLE SPACE** is used to store long or LOB column data. It can also store structured type columns or index data.
- **SYSTEM TEMPORARY TABLE SPACE** is used to store temporary data such as intermediate tables during sort operations, reorganizing tables and indexes, joining tables or creating indexes. A temporary table space is created automatically when the database is created.
- **USER TEMPORARY TABLE SPACE** is used to store declared global temporary tables (DGTT) for the life of a session.

These table spaces may be defined as using a System Managed Space (SMS) or a Database Managed Space (DMS).

- In an SMS table space, the operating system's file system manager allocates and manages the space where the table is stored. The storage model typically consists of many files, representing table objects, stored in the file system space. The user decides on the location of the files, DB2 controls their names, and the file system is responsible for managing them.

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

  In an SMS table space, a file is extended one page at a time as the object grows, with extent-sized allocations like DMS table spaces. However, this can be changed to multipage file allocation by executing the **db2empfa** utility to enhance the performance of workloads with high volumes of inserts. The informational database configuration parameter **multipage_alloc** is set to YES to reflect the fact that the default behavior has been changed.

  Note: multipage_alloc applies to *all* SMS table spaces in the database, and once it has been enabled, it cannot be reversed.

SMS space usage can be monitored through operating system commands.

- In a **DMS** table space, the database manager controls the storage space. The storage model consists of a limited number of devices or files whose space is managed by DB2. The DBA decides which devices and files to use, and DB2 manages the space on those devices and files. The table space is essentially an implementation of a special purpose file system designed to best meet the needs of the database manager.

  A DMS table space containing user-defined tables and data can be defined as:

  – A regular table space to store any table data and optionally index data
  – A large table space to store long field or LOB data or index data
  – A temporary table space

For further details on SMS and DMS table spaces, refer to *DB2 UDB Administration Guide: Planning*, SC09-4822.

Performance considerations

The following considerations apply to table space design:

- SMS or DMS table space?
- Number of tables per table space and table spaces per table?
- Choosing table space containers
- How many containers?
- Page size for the table space?
- Extent size for the table space?
- Prefetch size for the table space?

These considerations are discussed in the following subsections.

SMS or DMS table space

A brief review of the advantages and disadvantages of SMS and DMS table spaces follows:

- **SMS**

  Some of the *advantages* of SMS table spaces are:

  – Space is not allocated by the system until it is required.

  – Easier to manage; creating a table space requires less initial work, because the containers do not have to be predefined.

  Some of the *disadvantages* of SMS table spaces are:

  – Containers cannot be added or deleted after the table space has been created.

  – Table space is considered full as soon as one of the containers is filled.

  - Indexes cannot be assigned to a table space different from the one its table is associated with, and therefore assigned to a different buffer pool.

- **DMS**

  Some of the *advantages* of DMS table spaces are:

  - The size of a table space can be increased (by adding or extending containers) or decreased (by reducing the size of a container, or dropping a container). The `ALTER TABLESPACE` statement is used to accomplish this task. Existing data can be automatically rebalanced across the new set of containers to retain optimal I/O efficiency.
  - A table can be split across multiple table spaces, based on the type of data being stored:
    - Long field and LOB data
    - Indexes
    - Regular table data

    This gives greater tuning flexibility such as spreading the data across multiple disks and assigning different buffer pools for indexes, as well as increasing/decreasing the amount of space available for a table.

  Some of the *disadvantages* of DMS table spaces are:

  - Container space has to be preallocated
  - Administration is complex
  - Incorrect allocation cannot be rectified as easily

Best practices

We recommend the following best practices for SMS/DMS design:

1. In general, a well-tuned set of user data DMS table spaces will outperform SMS table spaces. Therefore, DMS is a good choice if performance is critical.

   Note: The exception to this is that SMS generally outperforms DMS for temporary table spaces, except perhaps when large sorts are involved.

2. SMS table spaces are recommended when small-to-moderate growth is expected, performance requirements are not as stringent, and there is a dearth of DBA skills to administer the database. Small personal databases are easiest to manage with SMS table spaces.
3. For large, growing tables where performance is critical, DMS table spaces with multiple containers per table are preferred since they offer larger sizes and greater tuning flexibility such as separation of data and indexes over multiple physical disks and buffer pools, use of RAW devices for performance gain, and separation of regular and long data, if desired.

When in doubt, and performance is the overriding factor, use DMS table space raw logical volumes.

Note: DB2 UDB Version 8 offers the ability to add a DMS container such that a rebalance does not occur. The `BEGIN NEW STRIPE SET` option of the `ALTER TABLESPACE` statement permits a container to be added above the high water mark so that a rebalance does not occur.

DB2 UDB Version 8 also added the ability to drop or shrink DMS containers.

Number of tables per table space and table spaces per table?

The *advantages* of multiple tables per table space are essentially ease of administration and space savings for small tables, while the potential *disadvantages* are performance degradation due to disk contention accessing different tables in the same table space on the same device at the same time, and point-in-time recovery of individual tables.

Consider sharing a table space among tables under the following circumstances:

1. A collection of tables related through system or user-defined referential integrity (RI) may share the same table space, if frequent point-in-time recoveries are performed. This ensures that all related tables are consistent after the recovery, thus avoiding any missed table spaces that could result in `CHECK PENDING` states.
2. A number of small tables with small amounts of data that change infrequently can share the same table space.
3. Group infrequently accessed data together in table spaces that use containers on slower devices.

The *advantages* of a single table per table space or multiple table spaces per table are greater tuning flexibility through separation of data and indexes for placement and buffer pool assignment, and more granularity in defining individual table backup strategies based on data volatility. The potential *disadvantages* are the proliferation of table spaces and consequent administration challenges, such as ensuring consistent point-in-time recovery of a collection of table spaces.

Consider single table per table space or multiple table spaces per table under the following circumstances:

1. With DMS table spaces, spanned tables can be defined (an example of such a table is one which has its data in one table space, indexes in another table space, and LOB data in a third table space).

Spanned tables offer greater tuning flexibility such as separating data and indexes, and placing the indexes in a table space with its own dedicated buffer pool; this can help ensure that index pages are kept in memory.

2. Larger base tables which are heavily accessed and frequently updated justify having their own SMS/DMS table space, and so do important tables. Isolating tables in their own table space provides greater tuning flexibility since they can be placed on separate disks and assigned a separate buffer pool. It also provides greater flexibility in devising backup strategies based on the volatility of individual table spaces.
3. If tables need to be monitored individually using commands such as the `LIST TABLESPACE SHOW DETAIL` command, then those tables must have their own table space. This command lists the number of used pages. In the case of a DMS table space, the command also reports the number of free pages and high water mark information.
4. For very large tables, consider table space capacity limits in isolating each table in its own table space. The limits for single partition DB2 UDB table spaces are 64 GB (4 K pages), 128 GB (8 K pages), 256 GB (16 K pages), 512 GB (32 K pages), 2 TB for long table spaces.

Choosing table space containers

A *container* is an allocation of physical storage such as a file or a device, and can be identified by a directory name, a device name, or a file name. Containers are assigned to table spaces. A single table space can span many containers, but each container can belong to only one table space.

Typically disk I/O throughput with device containers (raw logical volumes on AIX) shows an improvement of 10% to 35% as compared to file systems (JFS file systems on AIX); the upper limit tends not to apply to databases. However, the DBA needs to take into account that actual gains depend on the I/O workload mix of the particular application.

A brief review of the considerations associated with each container type follows:

- Directory containers can only be exploited by SMS table spaces.
- File containers are used by DMS table spaces and are files of a preallocated size.

  DMS treats file and device containers in the same way. When the file is created, DB2 will allocate the entire container (file) at table space creation time. Even though this allocation is done at creation time, the operating system's file system may still fragment the file, resulting in the allocation of pages being non-contiguous.
- Device containers can only be used by DMS table spaces.

As in the case of file containers, space is also allocated at table space creation time. However, unlike file containers, DB2 interacts with the raw device directly, which ensures that page allocation is contiguous.

DMS table spaces that use raw device containers generally perform better than file containers. The reason for this is that with file containers, the database manager interacts with the operating system's file system layer, unlike raw device containers which the database manager manages directly. An exception to this rule is a DMS table space using file containers that hold LOB data. In this case, file container performance might be greater than that of a device container, since the database manager takes advantage of the file system cache.

Note: LOB and LONG VARCHAR data is *not* buffered by DB2.

Table 3-3 summarizes the relationship between containers DMS/SMS table spaces.

Table 3-3 Container types for DMS and SMS table spaces

Container	DMS	SMS
Directory		Yes
File	Yes	
Device	Yes	

Best practices

We recommend the following best practices for choosing containers for table spaces:

1. Directory containers only apply to SMS table spaces. Ensure that all directories (containers) are specified at table space creation, since they cannot be added later.

   Each container of a table space should be mapped to one or more different physical drives to improve parallel I/O.

2. Device containers should be used when maximum performance is desired; the exception being DMS table spaces designed for LOB data, which would benefit from file containers.

   The use of DMS device containers avoids the double buffering scenario which can occur with DMS file containers or SMS directory containers. This occurs when pages are cached by the operating system in its file system cache, and by DB2 in the buffer pools.

3. Workloads like OLTP, which perform a large amount of random I/O operations, benefit from the use of raw logical volumes.
4. Customers with BI applications, which perform a large number of sequential I/O operations, generally benefit from the sequential read-ahead feature of JFS file systems.

How many containers

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

Data for any table will be stored on all containers in a table space in a round-robin fashion, as shown in Figure 3-8. It depicts how DB2 writes extent 0 to container 1 on Disk A, then extent 1 to container 2 on Disk B, and then extent 2 to container 3 on Disk C, and then continues in a round-robin fashion writing extents 3, 4, and 5. This balances the data across the containers that belong to a given table space. The number of pages that the database manager writes to one container before using a different one is called the *extent size*.

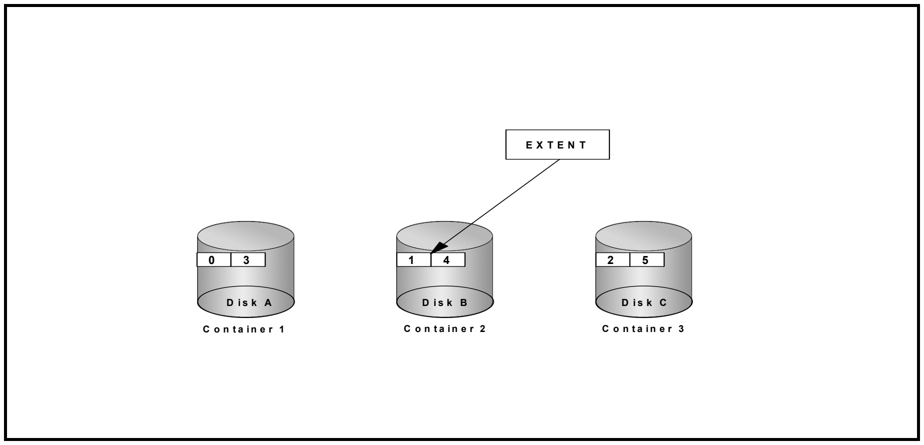


Figure 3-8 DB2 striping with containers

The number of containers defined for a table space has an impact on access performance, and can limit the maximum size achievable for a table space.

- The size of a table space can be limited by the number of containers defined. Each table space is limited to a maximum size for a given page size; for example, a 4 K page size limits a table space to 64 GB, while an 8 K page size limits it to 128 GB.

  However, a poor choice in the number of containers can further restrict the size of a table space to less than the maximum allowed for a given page size. For example, assuming each container is mapped to a separate file system, the maximum achievable size of the table space (within the ceiling for a given page size as mentioned earlier) is limited to the number of containers multiplied by the maximum file system size supported by the operating system.

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

  While multiple containers can be created on the same physical disk, each container for a table space should use a different disk for superior parallel I/O performance.

The following considerations apply to containers.

Single or multiple containers

The size of the table space vis-a-vis the available container size determines whether or not a single container may be used for the table space.

When the table space exceeds the size of a single container, an appropriate number of containers needs to be defined.

In general, even small table spaces that could easily fit in a single container can benefit from having multiple containers spread over a number of physical volumes.

Attention: Multiple containers per file system may also be appropriate to reduce inode contention in AIX.

Best practices

Assuming that an adequate number of disks are available, we recommend that each container be assigned to a separate disk or file system to maximize space availability, minimize contention and improve performance, as follows:

1. Assigning each disk or file system to a table space provides the maximum space capacity available to it.
2. Separating tables scanned simultaneously into separate table spaces with separate disks can reduce disk head movement (and thus, seek time) when doing sequential scans.
3. Any disk errors will be isolated to a single table space.
4. For OLTP environments with predominantly random I/O, consider having each “disk” (single spindle or RAID array) spanned by multiple table spaces in order to minimize hot-spots.

Size of containers

With single container table spaces, the container size limits the volume of data stored.

With multiple containers per table space, the size of the individual containers can impact parallel I/O performance as well as space utilization across all the containers as follows:

- With unequal size containers and SMS table spaces, the table space is considered full as soon as any *one* of the containers becomes full (this does not apply to DMS table spaces).
- With unequal sizes of containers for a given DMS table space, the effectiveness of any parallel prefetches performed against that table space is reduced.
  - When a smaller container becomes full and the data gets written to one of the larger containers, the data in the containers become unbalanced. This imbalance considerably reduces the effectiveness of parallel prefetching, since a query may require only a small portion of the data from the small container and a much larger portion from the larger container.
  - When a small container is added to a table space which contains a number of large containers, rebalance may occur (depending upon the `BEGIN NEW STRIPE SET` option), which could result in the small container containing far less data than that in the large containers, which will again lead to less optimal parallel prefetching.

Best practices

We recommend that all containers for a single table space have the same size to enhance parallel prefetching (for DMS table spaces) and avoid wasting space (SMS table spaces).

Page size for the table space

The page size of a table space can affect the performance of an application and overall system performance, as follows:

- Too small a page size can impact the performance of sequential processing due to multiple I/Os. It may also increase the number of levels in the index for large index key sizes and numbers of rows in the table.

  However, smaller page sizes result in less page contention for update workloads, reduce or eliminate wasted space in a page when using small rows, avoid wastage of space in the buffer pool, and have smaller I/O request sizes.
- Large page sizes can consume unnecessary memory in the buffer pools with random I/Os, thereby impacting overall system performance. However, large

page sizes increase a table space’s capacity, and enhance I/O performance for large data scans as more rows are read with each I/O operation.

Attention: A maximum of 255 rows can be accommodated on a page. Choosing a page size greater than this limit only results in wasted space in the page and memory in the buffer pool.

Table 3-4 summarizes the limits in rows sizes and number of columns for a given page size.

Table 3-4 Page size, row and column limits

Page size	Maximum row length	Maximum columns
4 KB	4,005 bytes	500
8 KB	8,101 bytes	1012
16 KB	16,293 bytes	1012
32 KB	32,677 bytes	1012

Attention: Once the page size is defined for a table space, it cannot be altered.

Best practices

We recommend the following best practices for the page size of a table space:

1. For OLTP workloads that mainly perform random read and writes, choose smaller page sizes to ensure no wastage of space in the buffer pools.
2. For BI types workloads that perform sequential processing, choose larger page sizes within the constraints of a maximum of 255 rows per page.
3. Consider larger page sizes for indexes in separate table spaces to reduce the number of index levels, since more index keys will fit on each index leaf page. However, since each index page can have only 255 index entries, a small index size could result in wasted space with a large page size.

Extent size for the table space

The extent size (`EXTENTSIZE` parameter in the `CREATE TABLESPACE` statement) determines the number of pages written to one container before switching to the next container in round-robin fashion. It is also the unit of space allocation with DMS table spaces. With SMS table spaces, the allocation is a page at a time, unless multipage allocation is enabled via the `db2empfa` utility. The extent size is also used in conjunction with the prefetch size to improve the performance of sequential access as described in “Prefetch size for the table space” on page 152.

The default extent size is specified by the **DFT_EXTENT_SZ** database configuration parameter, which has the default value of 32 pages.

Attention: Once the extent size is defined for a table space, it cannot be altered.

DB2 allocates a minimum of two extents for each table object. Figure 3-9 highlights the extent allocation for DMS table spaces.

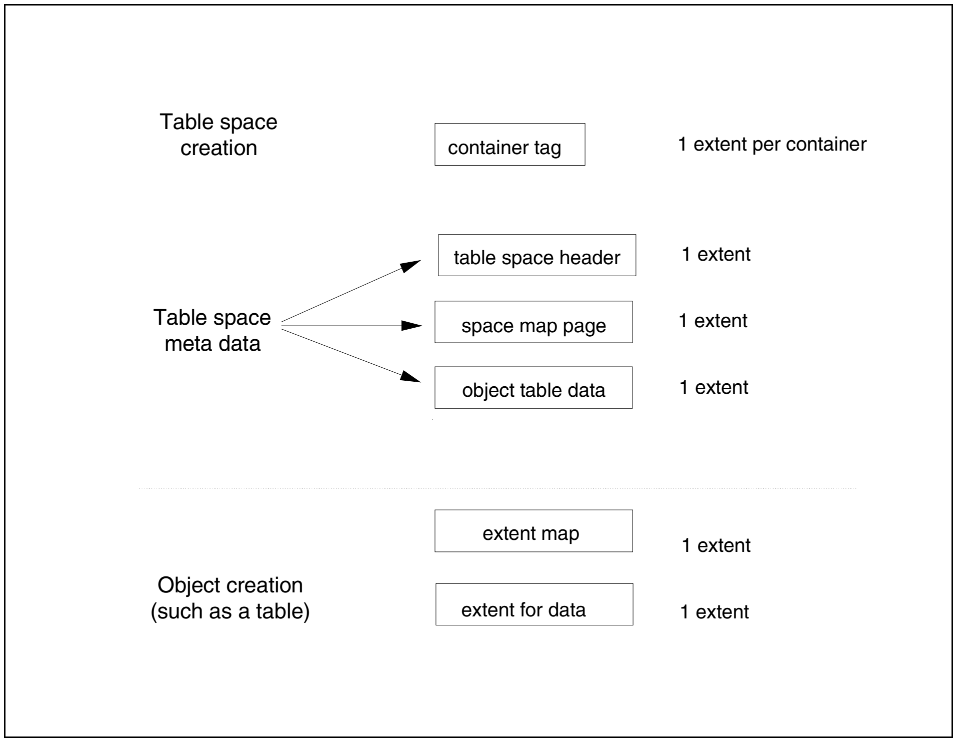


Figure 3-9 Extent allocation for DMS table space

The minimum initial extents requirement for a DMS table space when the first storage object is created in it is as follows, assuming the default extent size of 32 pages:

```
6 extents = (6 * 32) = 192 pages
```

Note: By default in DB2 UDB Version 8, 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. Prior to 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 the new registry variable `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 this registry variable, you need to issue a `db2stop` command followed by a `db2start` command.

Best practices

We recommend the following best practices in the selection of the extent size:

1. The default extent size is generally acceptable.
2. When using striped containers, the extent size should be an integral multiple of the stripe size (such as RAID strip size).
3. Reduce the extent size for DMS table spaces with multiple small tables to save space. If performance is not critical for these tables, consider using SMS table spaces which allocate a page at a time.
4. For large tables in DMS table spaces with high insert activity, consider increasing the extent size, since smaller extents will result in additional overhead to allocate frequently required extents.
5. Use larger extent size values for table spaces that are mostly accessed in a sequential manner as in BI environments. This would positively affect sequential prefetching performance, as more data would be processed per I/O request.

Prefetch size for the table space

DB2 may use sequential prefetch when accessing data sequentially; it issues asynchronous sequential prefetch requests to bring data into the buffer pool in advance of the application needing it. DB2 anticipates the sequential nature of an application's process such as a table scan, and enhances the performance of the application by having the data in the buffer pool instead of necessitating synchronous application I/O. DB2 recognizes an application's sequential processing characteristics at bind time for static SQL, or prepare time for dynamic SQL, and incorporates sequential prefetch into the access plan developed.

DB2 also supports sequential detection, which detects data being accessed in a sequential manner at run-time, which also triggers sequential prefetch.

The prefetch size (**`PREFETCHSIZE`** parameter in the **`CREATE TABLESPACE`** statement), the extent size, and the container layout on disk are important factors in determining prefetch performance and consequently the overall performance of the database. When developing a prefetch request, DB2 considers the extent size, prefetch size and the number of containers to develop the appropriate I/O requests. If a table space's containers are allocated over separate physical disks (disk actuators or arms), DB2 can issue parallel read requests to improve performance.

Note: The buffer pool manager component may reject sequential prefetch requests if it determines that pages that are currently used, or likely to be used, by executing applications would be invalidated by the pages coming in. Thus, aggressive prefetching can hurt performance instead of helping it.

If multiple containers are used, DB2 can issue multiple extent size prefetch requests. For example, with a prefetch size of 96 pages, an extent size of 32 pages, and three containers, DB2 will issue three extent size prefetch requests against the three containers. If the three containers are allocated over separate physical disks or ranks in the case of IBM ESS, parallel I/O will be used using multiple prefetchers.

Figure 3-10 on page 154 shows how DB2 can drive multiple prefetch requests with a table space created on RAID-5 with the **`DB2_PARALLEL_IO`**[3] registry variable enabled for all table spaces in the database.

[3] When this variable is enabled, the database manager is able to issue parallel I/O requests to the table space, even though it consists of a single container. This is appropriate for table spaces created on RAID devices that are made up of more than one physical disk. When this variable is disabled, the number of prefetcher requests created is based on the number of containers in the table space.

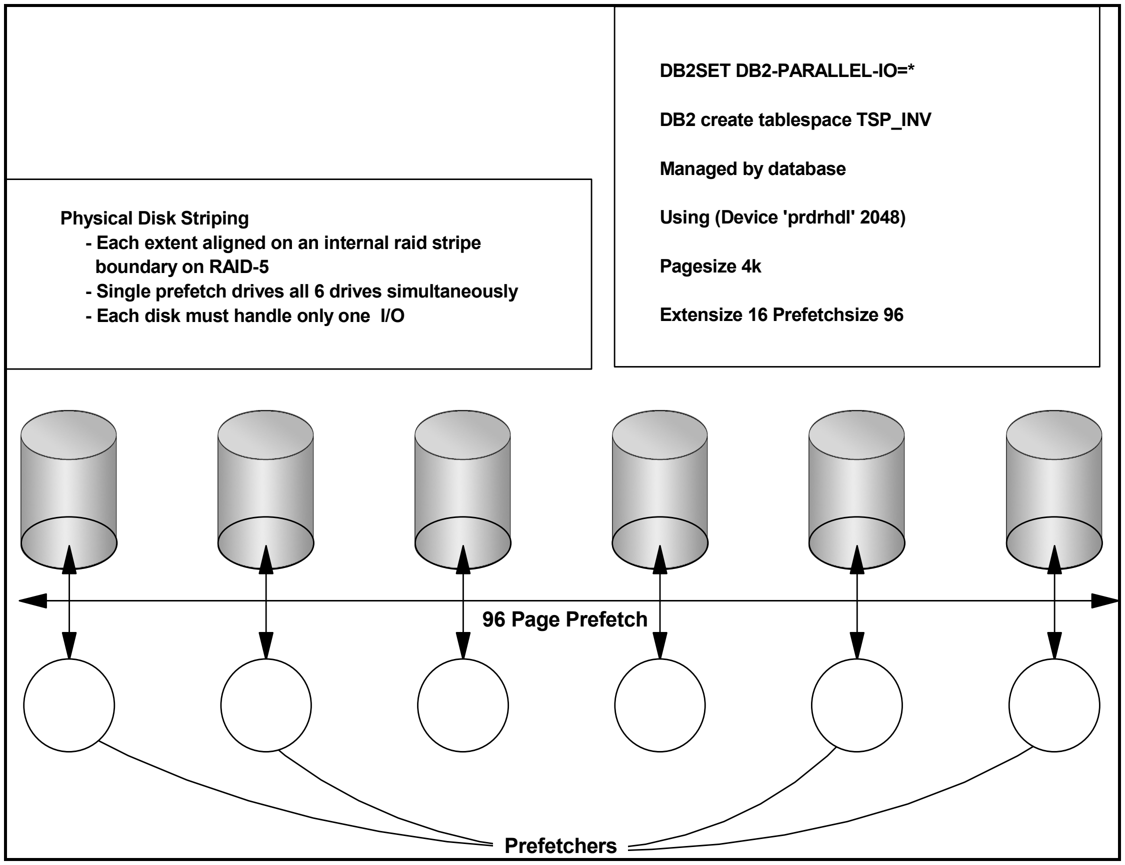

DB2SET DB2-PARALLEL-IO=*
DB2 create tablespace TSP_INV
Managed by database
Using (Device 'prdrhdl' 2048)
Pagesize 4k
Extensize 16 Prefetchsize 96
Physical Disk Striping
- Each extent aligned on an internal raid stripe boundary on RAID-5
- Single prefetch drives all 6 drives simultaneously
- Each disk must handle only one I/O
96 Page Prefetch
Prefetchers

Figure 3-10 DB2 prefetch example using RAID-5

Setting the correct prefetch size correctly will improve the database manager's ability to read pages in advance and reduce execution times.

The default value for prefetch is defined by the database configuration parameter **`DFT_PREFETCH_SZ`**. Prefetch can be disabled by setting **`PREFETCHSIZE`** to zero for this table space. The prefetch size can be modified via the **`ALTER TABLESPACE`** statement.

Best practices

We recommend the following best practices for choosing the prefetch size:

1. The default prefetch size (32 pages on UNIX, 16 on Windows) generally delivers acceptable performance.

2. For optimal parallel prefetching, the containers in the table space should reside on separate physical devices. This is particularly important when using a DMS raw container, as there will be no prefetching or caching performed by the operating system.
3. The prefetch size should be an integral multiple of **`EXTENTSIZE`**.

   For most environments, choose a prefetch size that is equal to (**`EXTENTSIZE`** * number of containers).

Performance monitoring metrics

The number of pages prefetched per request should be close to or equal to the extent size. If the number of pages prefetched per request in to the buffer pool are less than the extent size, the following conditions may exist:

- Buffer pool overheated/constrained
- Extent size is too large
- Prefetch size too large

3.3.6 Writing efficient SQL

SQL is a high level language that provides considerable flexibility in writing queries which deliver the same answer set. However, not all forms of the SQL statement deliver the same performance for a given query. It is therefore vital to ensure that the SQL statement is written in a manner to provide optimal performance.

DB2 UDB provides the SQL compiler which creates the compiled form of SQL statements. When the SQL compiler compiles SQL statements, it rewrites them into a form that can be optimized more easily. This is known as *query rewrite*. The SQL compiler then generates many alternative execution plans for satisfying the user's request as shown in Figure 3-11 on page 156.

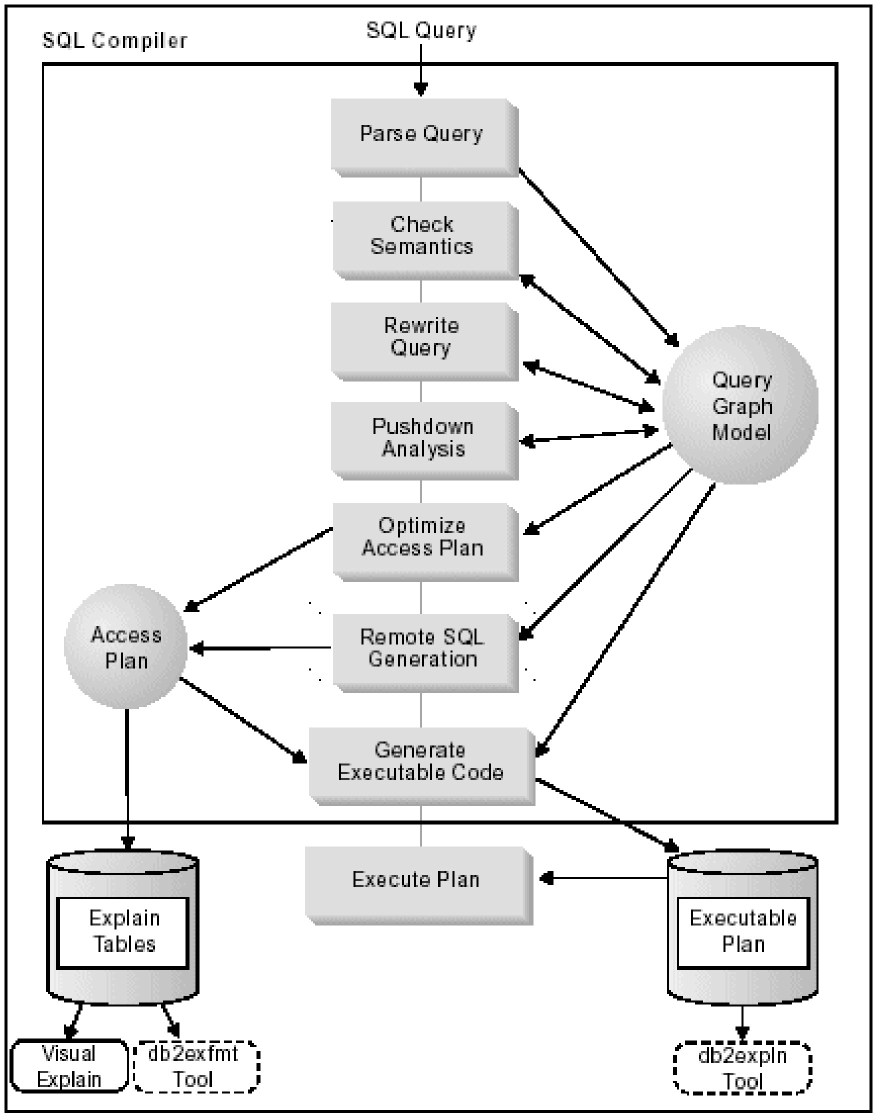


Figure 3-11 The stages of the SQL compiler

The SQL compiler estimates the execution cost of each alternative plan using the statistics for tables, indexes, columns, and functions, and chooses the plan with the smallest execution cost. This is known as *query optimization*.

The access plan for static SQL statements is created at bind time and contained in a package. The package contains the access plan for all SQL statements in the application program. For dynamic SQL statements, the access plan for each SQL statement is created when the application is executed.

The following is a brief review of the main steps performed by the SQL compiler as described in Figure 3-11 on page 156.

1. **Parse Query** involves analyzing the SQL query to check the syntax. When parsing query is completed, an internal representation of the query called the *query graph model* is created.
2. **Check Semantics** involves checks such as ensuring that the data type of the column specified for the **AVG** column function is a numeric data type, determining whether the views in the query need to be merged or materialized.
3. **Rewrite Query** involves the SQL compiler rewriting the query to make it more efficient, while retaining the semantic equivalence of the original query. Categories of query rewrite include the following:
   - Operation Merging, where a subquery can sometimes be merged into the main query as a join, giving the optimizer more choices to determine the most efficient access plan.
   - Operation Moment, where the optimizer may remove the **DISTINCT** operation to reduce the cost of operation.
   - Predicate translation, where the SQL compiler may rewrite queries to translate existing predicates to more optimal predicates.
4. **Federated Pushdown Analysis** is bypassed unless it is a federated database query.
5. **Optimize Access Plan** involves creating an executable access plan, or section, for the query. Information about the access plan for static SQL is stored in the system catalog tables, which is used by the **db2expln** explain tool.
6. **Remote SQL Generation** relates to 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 SQL dialect of the data source.
7. **EXPLAIN Tables** is where access path details are stored when EXPLAIN is invoked. Either GUI tool Visual Explain or a text-based **db2exfmt** tool can be used to examine the explain information in these tables.

Important: Keeping the database objects' statistics up-to-date is critical to the DB2 optimizer choosing the optimal access plan.

From an application programmer’s perspective, coding efficient SQL involves the following considerations:

1. Dynamic or static SQL
2. Minimizing the number of SQL statements issued
3. Limiting the volume of data returned - columns and rows
4. Guiding the DB2 optimizer towards a superior access path
5. Avoiding data type conversions
6. Minimizing network overheads
7. Minimizing concurrency problems

Dynamic or static SQL

When SQL statements are embedded in a program, they are called embedded SQL programs. SQL can be embedded in C/C++, COBOL, and Java (SQLJ) programming languages.

There are two types of SQL statements: dynamic and static. A brief overview of each follows:

- **Dynamic SQL** statements are ones where the application builds and executes the SQL at run time.

  An interactive application that prompts the end user for key parts of an SQL statement, such as the names of the tables and columns to be searched, is a common example of dynamic SQL. The application builds the SQL statement at runtime, and then submits the statement for processing.

  Since access path generation occurs at run time (inline with query execution), dynamic SQL statements leverage current database statistics in generating an optimal access plan. However, dynamic SQL statements can take more time to execute, since queries are optimized at runtime.

  Performance of dynamic SQL statements can be improved by minimizing preparation time by tuning the query optimization level.

  If the query optimization class is higher than necessary, the preparation times can be high since the DB2 Optimizer can spend more time finding the best access plan for a query than is justified by a reduction in execution time.

- **Static SQL** statements are ones where the SQL statement type and the database objects accessed by the statement, such as column names, are known prior to running the application. The only unknowns are the data values the statement is searching for or modifying.

  Applications using static SQL must be precompiled and bound to the database, so that the database manager analyzes all the static SQL statements in a program, determines its optimal access plan, and stores the ready-to-execute package prior to the execution of the program. Since all the logic required to execute SQL statements is determined before executing the program, static SQL programs have the least run time overhead of all DB2 programming methods, and execute faster.

Best practices

We recommend the following best practices for choosing between dynamic and static SQL:

1. Static SQL statements are well-suited for OLTP environments that demand high throughput and very fast response times. Queries tend to be simple, retrieve few rows and stable index access is the preferred access path.

   Note: Keeping table and index statistics up-to-date helps the DB2 optimizer choose the best access plan. However, SQL packages need to be rebound for the DB2 optimizer to generate a new access plan based on these statistics.

2. Dynamic SQL statements are generally well-suited for applications that run against a rapidly changing database, where queries need to be specified at run time. This is typical of BI environments.

   Dynamic SQL can be written to use literals or parameter markers, as shown in Example 3-3.

Example 3-3 Dynamic SQL with literals and parameter markers

```
---Dynamic SQL using literals
strcpy (st1,"SELECT * FROM EMPLOYEE WHERE empno=‘000100’");
strcpy (st2,"SELECT * FROM EMPLOYEE WHERE empno=‘000200’");
EXEC SQL PREPARE s1 FROM :st1;
EXEC SQL PREPARE s2 FROM :st2;
EXEC SQL DECLARE c1 CURSOR FOR s1;
EXEC SQL DECLARE c2 CURSOR FOR s2;
EXEC SQL OPEN c1;
EXEC SQL OPEN c2;
...
---Dynamic SQL using parameter markers
strcpy (st,"SELECT * FROM EMPLOYEE WHERE empno=’?’");
EXEC SQL PREPARE s1 FROM :st;
EXEC SQL DECLARE c1 CURSOR FOR s1;
EXEC SQL DECLARE c2 CURSOR FOR s1;
strcpy (parmvar1,"000100");
strcpy (parmvar2,"000100");
EXEC SQL OPEN c1 using :parmvar1;
...
EXEC SQL OPEN c2 using :parmvar2;
...
```

If literals are used, each time the statement is run using a new value for the literal requires DB2 to do a **PREPARE**. CPU time must be used to do the **PREPARE** and in high volume applications, this can be quite expensive.

A parameter marker is represented with a question mark (?) in place of the literal in the SQL statement. The parameter marker is replaced with a value at run time by the application. Therefore, the SQL statement can be reused from the package cache and does not require a subsequent **PREPARE**. This results in faster query execution and reduced CPU consumption.

Dynamic SQL is appropriate for OLTP environments as well. When used in OLTP environments in particular, we strongly recommend the use of parameter markers in dynamic SQL to achieve superior performance.

Note: Keeping table and index statistics up-to-date helps the DB2 optimizer choose the best access plan. However, unlike the case with static SQL, packages with dynamic SQL do *not* need to be rebound after new indexes have been added and/or new statistics have been gathered. But the package cache needs to be flushed via the **FLUSH PACKAGE CACHE** command to ensure that the new statistics are picked up.

3. For OLTP environments characterized by high concurrent activity, simple SQL statements and subsecond response time requirements, the optimization class should be set (**SET CURRENT QUERY OPTIMIZATION** statement) to a lower value such as 1 or 2. If the optimization level is not set in the **CURRENT QUERY OPTIMIZATION** special register, the DB2 optimizer will table the value set in the **DFT_QUERYOPT** database configuration parameter.

Minimize the number of SQL statements issued

Avoid using multiple SQL statements when the same request can be issued using one SQL statement. This minimizes the cost of accessing DB2, and also provides more information in an SQL statement, which enables the DB2 optimizer to choose a more optimal access path.

Limit the volume of data returned - columns and rows

SQL performance is enhanced by specifying only the columns of interest in the select list of a query, and limiting the number of rows accessed using predicates.

Avoid the "**SELECT ***" construct, which specifies that all columns are to be returned in the result set, resulting in needless processing.

Figure 3-12 on page 161 shows a simplified version of the components in DB2 and the categories of predicates that they evaluate.

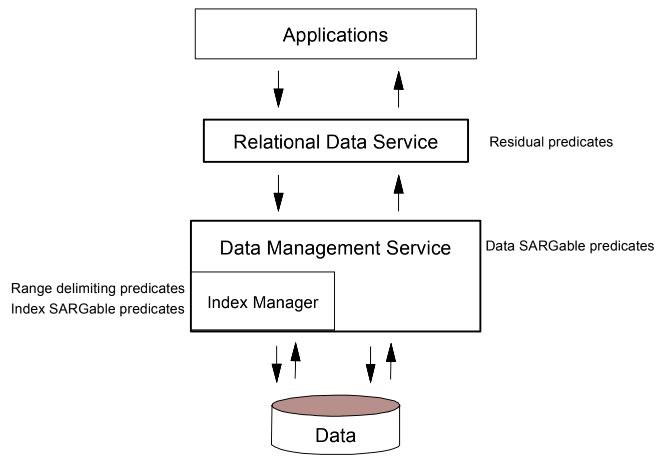


Figure 3-12 DB2 predicate types and components where they are evaluated

A brief description of the various components shown in Figure 3-12 follows:

- Relational Data Services (RDS) receive SQL requests from applications and return the result set. RDS supports many operations for processing data such as joins, grouping, aggregation, and math. RDS includes the DB2 optimizer, which performs access path optimization and generates compiled SQL sections. The RDS also includes catalog services that manage the metadata describing the tables, indexes and views, as well as statistics on the data.
- Data Management Services (DMS) provide the data structures for tables, long fields, large objects, and indexes. DMS also performs insert, update, and delete operations on these structures.

  Some of the components included in DMS are the Table Manager, Index Manager, Long object Manager and Large object Manager.

There are four categories of predicates, each with its own processing cost. The category is determined by how and when that predicate is evaluated.

The following categories list is ordered in terms of *performance*, starting with the most favorable:

1. Range delimiting predicates are those used to bracket an index scan; they provide start and/or stop key values for the index search. These predicates are evaluated by the Index Manager.
2. Index SARGable[4] predicates are not used to bracket a search, but are evaluated from the index if one is chosen, because the columns involved in the predicate are part of the index key. These predicates are also evaluated by the Index Manager.
3. Data SARGable predicates are the predicates that cannot be evaluated by the Index Manager, but can be evaluated by DMS. Typically, these predicates require the access of individual rows from a base table. If necessary, DMS will retrieve the columns needed to evaluate the predicate, as well as any others to satisfy the columns in the **SELECT** list that could not be obtained from the index.
4. Residual predicates are those that require I/O beyond the simple accessing of a base table. Examples of residual predicates include those using quantified sub-queries (sub-queries with **ANY, ALL, SOME**, or **IN**), or reading **LONG VARCHAR** or large object (LOB) data which is stored separately from the table. These predicates are evaluated by RDS and are the most expensive of the four categories of predicates.

Table 3-5 on page 163 provides examples of various predicates and identifies their type based on the context in which they are used.

Note: In the examples, we assume that a multi-column ascending index exists on (c1,c2,c3) and is used in evaluating the predicates where appropriate.

If any column in the index is in descending order, then the start and stop keys might be switched for range delimiting predicates.

[4] SARGable refers to the fact that something can be used as a search argument

Table 3-5 Predicate processing for different queries

Predicates	**Column c1**	**Column c2**	**Column c3**	**Comments**
c1 = 1 and c2 = 2 and c3 = 3	Range delimiting (start-stop)	Range delimiting (start-stop)	Range delimiting (start-stop)	The equality predicates on all the columns of the index can be applied as start-stop keys.
c1 = 1 and c2 = 2 and c3 >= 3	Range delimiting (start-stop)	Range delimiting (start-stop)	Range delimiting (start)	Columns c1 and c2 are bound by equality predicates, and the predicate on c3 is only applied as a start key.
c1 >= 1 and c2 = 2	Range delimiting (start)	Range delimiting (start-stop)	Not applicable	The leading column c1 has a >= predicate and can be used as a start key. The following column c2 has an equality predicate, and therefore can also be applied as a start-stop key.
c1 = 1 and c3 = 3	Range delimiting (start-stop)	Not applicable	Index SARGable	The predicate on c3 cannot be used as a start stop key, since there is no predicate on c2. It can, however, be applied as an Index SARGable predicate.
c1 = 1 and c2 > 2 and c3 = 3	Range delimiting (start-stop)	Range delimiting (start)	Index SARGable	The predicate on c3 cannot be applied as a start-stop predicate because the previous column has a > predicate. Had it been a >= instead, we would be able to use it as a start-stop key.
c1 = 1 and c2 <= 2 and c4 = 4	Range delimiting (start-stop)	Range delimiting (stop)	Data SARGable	Here the predicate on c2 is a <= predicate. It can be used as a stop key. The predicate on c4 cannot be applied on the index and is applied as a Data SARGable predicate during the FETCH.

Predicates	Column c1	Column c2	Column c3	Comments
c2 = 2 and UDF_with_external_action(c4)	Not applicable	Index SARGable	Residual	The leading column c1 does not have a predicate, and therefore the predicate on c2 can be applied as an Index SARGable predicate where the whole index is scanned. The predicate involving the user-defined function with external action is applied as a Residual predicate.
c1 = 1 or c2 = 2	Index SARGable	Index SARGable	Not applicable	The presence of an OR does not allow us this multi-column index to be used as start-stop keys. This might have been possible had there been two indexes, one with a leading column on c1, and the other with a leading column on c2, and the DB2 optimizer chose an "index-ORing" plan. However, in this case the two predicates are treated as Index SARGable predicates.
c1 < 5 and (c2 = 2 or c3 = 3)	Range delimiting (stop)	Index SARGable	Index SARGable	Here the leading column c1 is exploited to stop the index scan from using the predicate with a stop key. The OR predicate on c2 and c3 are applied as Index SARGable predicates.

The DB2 optimizer employs the query rewrite mechanism to transform many complex user-written predicates into better performing queries, as shown in Table 3-6 on page 165.

Table 3-6 Query rewrite predicates

Original predicate or query	Optimized predicates	Comments
c1 between 5 and 10	c1 >= 5 and c1 <= 10	The between predicates are rewritten into the equivalent range delimiting predicates so that they can be used internally as though the user specified the range delimiting predicates.
c1 not between 5 and 10	c1 < 5 or c1 > 10	The presence of the OR predicate does not allow the use of a start-stop key unless the DB2 optimizer chooses an index-ORing plan.
select * from t1 where EXISTS (select c1 from t2 where t1.c1 = t2.c1)	select t1.* from t1 EOJOIN t2 where t1.c1= t2.c1	The subquery may be transformed into a join — internally a special join called an “early out join” is used so that we do not multiply the rows from t1 if there are multiple rows having the same value in the join column in t2
select * from t1 where t1.c1 IN (select c1 from t2)	select t1.* from t1 EOJOIN t2 where t1.c1= t2.c1	This is similar to the transformation for the EXISTS predicate example above.
c1 like 'abc%'	c1 >= 'abc X X X ' and c1 <= 'abc Y Y Y'	If we have c1 as the leading column of an index, DB2 generates these predicates so that they can be applied as range-delimiting start-stop predicates. Here the characters X and Y are symbolic of the lowest and highest collating character.
c1 like 'abc%def'	c1 >= 'abc X X X ' and c1 <= 'abc Y Y Y' and c1 like 'abc%def'	This is like the previous case, except that we have to additionally apply the original predicate as an index SARGable predicate so as to get the match for the characters “def” correctly.

Performance monitoring metrics

The explain tables identify the type of predicates in the predicate table. Example 3-4 shows a sample SQL statement before and after query rewrite has been performed by the DB2 optimizer. Assume there is an ascending multi-column index on LAST_NAME, FIRST_NAME, DEPARTMENT_NO, AGE, and that the DB2 optimizer uses this index in the execution of the query.

Example 3-4 Sample SQL statement before and after query rewrite

```
------------------
ORIGINAL STATEMENT:
------------------
select *
from employee
where last_name = 'ALUR' and first_name like 'N%' and department_no <> 490
        and age > 50 and salary > 100000.00 and 1 =
case
when exists
   (select 1
   from t1
   where employee.bonus = t1.c1)
then 1
when bonus > 10000
then 1
else 0 end

-----------------------------------------
OPTIMIZED STATEMENT (after query rewrite):
-----------------------------------------
SELECT Q3.SERIAL_NO AS "SERIAL_NO", Q3.FIRST_NAME AS "FIRST_NAME", 'ALUR ' AS
        "LAST_NAME", Q3.DEPARTMENT_NO AS "DEPARTMENT_NO", Q3.AGE AS "AGE",
        Q3.SALARY AS "SALARY", Q3.BONUS AS "BONUS", Q3.VARIABLE_PAY AS
        "VARIABLE_PAY"
FROM CALISTO.EMPLOYEE AS Q3
WHERE (+100000.00 < Q3.SALARY) AND (50 < Q3.AGE) AND (Q3.DEPARTMENT_NO <>
        490) AND (Q3.FIRST_NAME LIKE 'N%') AND (Q3.LAST_NAME = 'ALUR ') AND
        (1 =
CASE
WHEN EXISTS(SELECT $RID$
FROM CALISTO.T1 AS Q1
WHERE (Q3.BONUS = Q1.C1))
THEN 1
WHEN (Q3.BONUS > 10000)
THEN 1
ELSE 0 END )
```

The type of each predicate in the original query is as shown in Example 3-5.

Example 3-5 Predicate type of each predicate using the original query

```
last_name = 'ALUR'
This is a range delimiting predicate (start/stop)

first_name like 'N%'
This is a range delimiting predicate (start/stop) after being converted to
(Q3.FIRST_NAME >= 'NXXXXXXXXXXXXXXXXXXX')
(Q3.FIRST_NAME <= 'NYYYYYYYYYYYYYYYYYYY')
where X and Y are are symbolical of the lowest and highest collating character

department_no <> 490
This is an index SARGable preddicate

age > 50
This is an index SARGable preddicate since the previous column predicate was
not =, >=, <=

salary > 100000.00
This is a data SARGable predicate

1 = case when exists
            (select 1
             from t1
             where employee.bonus = t1.c1)
          then 1
          when bonus > 10000
          then 1
          else 0
     end
This is a residual predicate
```

When the original SQL statement is EXPLAINed, the DB2 optimizer generates entries in the various EXPLAIN tables corresponding to the transformed query (query rewrite), and the predicate table will reflect the transformed predicates and their type, as shown in Example 3-6 on page 168.

Note: The output does not differentiate between index SARGable and data SARGable predicates; it identifies both of them as SARGable predicates.

However, the operator defines whether the predicate is index SARGable or data SARGable—for example, the IXSCAN operator indicates an index SARGable predicate, while a TBSCAN indicates a data SARGable predicate.

Example 3-6 Predicate details from EXPLAIN

```
DB2 Universal Database Version 8.1, 5622-044 (c) Copyright IBM Corp. 1991, 2002
Licensed Material - Program Property of IBM
IBM DATABASE 2 Explain Table Format Tool

******************** EXPLAIN INSTANCE ********************

DB2_VERSION: 08.01.4
SOURCE_NAME: SQLC2E03
SOURCE_SCHEMA: NULLID
SOURCE_VERSION:
EXPLAIN_TIME: 2003-11-19-19.44.56.267596
EXPLAIN_REQUESTER: CALISTO

Database Context:
----------------
   Parallelism: Inter-Partition Parallelism
   CPU Speed: 4.000000e-05
   Comm Speed: 1.25
   Buffer Pool size: 1000
   Sort Heap size: 256
   Database Heap size: 1200
   Lock List size: 100
   Maximum Lock List: 10
   Average Applications: 1
   Locks Available: 850

Package Context:
---------------
   SQL Type: Dynamic
   Optimization Level: 5
   Blocking: Block All Cursors
   Isolation Level: Cursor Stability



---------------- STATEMENT 1  SECTION 203 ----------------
   QUERYNO: 1
   QUERYTAG:
   Statement Type: Select
   Updatable: No
   Deletable: No
   Query Degree: 1

....
....
....
Access Plan:
```

```
-----------
   Total Cost: 8.89956
   Query Degree:1

                  Rows
                 RETURN
                 (   1)
                  Cost
                   I/O
                   |
               0.00116053
                 DTQ
                 (   2)
                 8.89956
                 0.08704
                   |
               0.000580267
                 FETCH
                 (   3)
                 4.87761
                 0.08704
       +------------+------------+
     0.04352       4.8          34
     IXSCAN       BTQ      TABLE: CALISTO
     (   4)       (   5)      EMPLOYEE
     1.88854      42.2932
        0           1
       |            |
       34          2.4
 INDEX: CALISTO   TBSCAN
    EMP_INDEX     (   6)
                  37.3742
                    1
                   |
                   120
             TABLE: CALISTO
                   T1



   1) RETURN: (Return Result)
      Cumulative Total Cost: 8.89956
      Cumulative CPU Cost: 166804
      Cumulative I/O Cost: 0.08704
      Cumulative Re-Total Cost: 0.513165
      Cumulative Re-CPU Cost: 12829.1
      Cumulative Re-I/O Cost: 0
      Cumulative First Row Cost: 8.88862
```

```
		Cumulative Comm Cost:2.00007
		Cumulative First Comm Cost:0
		Estimated Bufferpool Buffers: 2.04352

		Arguments:
		---------
		BLDLEVEL: (Build level)
			DB2 v8.1.0.32 : n031022
		STMTHEAP: (Statement heap size)
			2048

		Input Streams:
		-------------
			8) From Operator #2

				Estimated number of rows: 0.00116053
				Partition Map ID:  -100
				Partitioning:      (COOR )
				            Coordinator Partition
				Number of columns: 8
				Subquery predicate ID: Not Applicable

				Column Names:
				------------
				+Q4.VARIABLE_PAY+Q4.BONUS+Q4.SALARY+Q4.AGE
				+Q4.DEPARTMENT_NO+Q4.LAST_NAME+Q4.FIRST_NAME
				+Q4.SERIAL_NO

				Partition Column Names:
				----------------------
				+NONE


	2) TQ    : (Table Queue)
		Cumulative Total Cost: 8.89956
		Cumulative CPU Cost: 166804
		Cumulative I/O Cost: 0.08704
		Cumulative Re-Total Cost: 0.513165
		Cumulative Re-CPU Cost: 12829.1
		Cumulative Re-I/O Cost: 0
		Cumulative First Row Cost: 8.88862
		Cumulative Comm Cost:2.00007
		Cumulative First Comm Cost:0
		Estimated Bufferpool Buffers: 2.04352

		Arguments:
		---------
		LISTENER: (Listener Table Queue type)
			FALSE
```

```
TQMERGE : (Merging Table Queue flag)
   FALSE
TQREAD  : (Table Queue Read type)
   READ AHEAD
TQSEND  : (Table Queue Write type)
   DIRECTED
UNIQUE  : (Uniqueness required flag)
   FALSE

Input Streams:
-------------
   7) From Operator #3

      Estimated number of rows: 0.000580267
      Partition Map ID:  1
      Partitioning:      (MULT )
                   Multiple Partitions
      Number of columns: 8
      Subquery predicate ID: Not Applicable

      Column Names:
      ------------
      +Q4.VARIABLE_PAY+Q4.BONUS+Q4.SALARY+Q4.AGE
      +Q4.DEPARTMENT_NO+Q4.LAST_NAME+Q4.FIRST_NAME
      +Q4.SERIAL_NO

      Partition Column Names:
      ----------------------
      +1: Q4.SERIAL_NO


Output Streams:
--------------
   8) To Operator #1

      Estimated number of rows: 0.00116053
      Partition Map ID:  -100
      Partitioning:      (COOR )
                   Coordinator Partition
      Number of columns: 8
      Subquery predicate ID: Not Applicable

      Column Names:
      ------------
      +Q4.VARIABLE_PAY+Q4.BONUS+Q4.SALARY+Q4.AGE
      +Q4.DEPARTMENT_NO+Q4.LAST_NAME+Q4.FIRST_NAME
      +Q4.SERIAL_NO

      Partition Column Names:
         ---------------------
         +NONE


3) FETCH : (Fetch)
   Cumulative Total Cost: 4.87761
   Cumulative CPU Cost: 66255.6
   Cumulative I/O Cost: 0.08704
   Cumulative Re-Total Cost: 0.513165
   Cumulative Re-CPU Cost: 12829.1
   Cumulative Re-I/O Cost: 0
   Cumulative First Row Cost: 4.86667
   Cumulative Comm Cost:0
   Cumulative First Comm Cost:0
   Estimated Bufferpool Buffers: 2.04352

   Arguments:
   ---------
   MAXPAGES: (Maximum pages for prefetch)
      1
   MAXPAGES: (Maximum pages for prefetch)
      1
   PREFETCH: (Type of Prefetch)
      NONE
   ROWLOCK : (Row Lock intent)
      NEXT KEY SHARE
   TABLOCK : (Table Lock intent)
      INTENT SHARE

   Predicates:
   ----------
   4) Sargable Predicate
      Relational Operator: Less Than (<)
      Subquery Input Required: No
      Filter Factor: 0.333333

      Predicate Text:
      --------------
      (+100000.00 < Q3.SALARY)

   9) Residual Predicate, Evaluate at Application Subquery
      Relational Operator: Equal (=)
      Subquery Input Required: Yes
      Filter Factor: 0.04

      Predicate Text:
      --------------
      (1 =
      CASE
   WHEN EXISTS(SELECT $RID$
   FROM CALISTO.T1 AS Q1
   WHERE (Q3.BONUS = Q1.C1))
   THEN 1
   WHEN (Q3.BONUS > 10000)
   THEN 1
   ELSE 0 END )


Input Streams:
-------------
   2) From Operator #4

      Estimated number of rows: 0.04352
      Partition Map ID:  1
      Partitioning:      (MULT )
                   Multiple Partitions
      Number of columns: 5
      Subquery predicate ID: Not Applicable

      Column Names:
      ------------
      +Q3.FIRST_NAME(A)+Q3.DEPARTMENT_NO(A)
      +Q3.AGE(A)+Q3.$RID$+Q3.LAST_NAME

      Partition Column Names:
      ----------------------
      +1: Q3.SERIAL_NO

   3) From Object CALISTO.EMPLOYEE

      Estimated number of rows: 34
      Partition Map ID:  1
      Partitioning:      (MULT )
                   Multiple Partitions
      Number of columns: 4
      Subquery predicate ID: Not Applicable

      Column Names:
      ------------
      +Q3.VARIABLE_PAY+Q3.SERIAL_NO+Q3.BONUS
      +Q3.SALARY

      Partition Column Names:
      ----------------------
      +1: Q3.SERIAL_NO

   6) From Operator #5
```

```
                Estimated number of rows: 4.8
                Partition Map ID:  1
                Partitioning:      (MULT )
                             Multiple Partitions
                Number of columns: 0
                Subquery predicate ID: 9

                Partition Column Names:
                ----------------------
                +NONE


          Output Streams:
          --------------
             7) To Operator #2

                Estimated number of rows: 0.000580267
                Partition Map ID:  1
                Partitioning:      (MULT )
                             Multiple Partitions
                Number of columns: 8
                Subquery predicate ID: Not Applicable

                Column Names:
                ------------
                +Q4.VARIABLE_PAY+Q4.BONUS+Q4.SALARY+Q4.AGE
                +Q4.DEPARTMENT_NO+Q4.LAST_NAME+Q4.FIRST_NAME
                +Q4.SERIAL_NO

                Partition Column Names:
                ----------------------
                +1: Q4.SERIAL_NO


       4) IXSCAN: (Index Scan)
          Cumulative Total Cost: 1.88854
          Cumulative CPU Cost: 47213.6
          Cumulative I/O Cost: 0
          Cumulative Re-Total Cost: 0.470103
          Cumulative Re-CPU Cost: 11752.6
          Cumulative Re-I/O Cost: 0
          Cumulative First Row Cost: 1.87918
          Cumulative Comm Cost:0
          Cumulative First Comm Cost:0
          Estimated Bufferpool Buffers: 1

          Arguments:
          ---------
          MAXPAGES: (Maximum pages for prefetch)
```

```
   ALL
PREFETCH: (Type of Prefetch)
   NONE
ROWLOCK : (Row Lock intent)
   NEXT KEY SHARE
SCANDIR : (Scan Direction)
   FORWARD
TABLOCK : (Table Lock intent)
   INTENT SHARE

Predicates:
----------
2) Stop Key Predicate
   Relational Operator: Less Than or Equal (<=)
   Subquery Input Required: No
   Filter Factor: 0.333333

   Predicate Text:
   --------------
   (Q3.FIRST_NAME <= 'NZZZZZZZZZZZZZZZZZZZ')

3) Start Key Predicate
   Relational Operator: Greater Than or Equal (>=)
   Subquery Input Required: No
   Filter Factor: 0.333333

   Predicate Text:
   --------------
   (Q3.FIRST_NAME >= 'N...................')

5) Sargable Predicate
   Relational Operator: Less Than (<)
   Subquery Input Required: No
   Filter Factor: 0.333333

   Predicate Text:
   --------------
   (50 < Q3.AGE)

6) Sargable Predicate
   Relational Operator: Not Equal (<>)
   Subquery Input Required: No
   Filter Factor: 0.96

   Predicate Text:
   --------------
   (Q3.DEPARTMENT_NO <> 490)

8) Start Key Predicate
```

```
   Relational Operator: Equal (=)
   Subquery Input Required: No
   Filter Factor: 0.04

   Predicate Text:
   --------------
   (Q3.LAST_NAME = 'ALUR ')

8) Stop Key Predicate
   Relational Operator: Equal (=)
   Subquery Input Required: No
   Filter Factor: 0.04

   Predicate Text:
   --------------
   (Q3.LAST_NAME = 'ALUR ')


Input Streams:
-------------
   1) From Object CALISTO.EMP_INDEX

      Estimated number of rows: 34
      Partition Map ID:  1
      Partitioning:      (MULT )
                   Multiple Partitions
      Number of columns: 5
      Subquery predicate ID: Not Applicable

      Column Names:
      ------------
      +Q3.FIRST_NAME(A)+Q3.DEPARTMENT_NO(A)
      +Q3.AGE(A)+Q3.$RID$+Q3.LAST_NAME

      Partition Column Names:
      ----------------------
      +1: Q3.SERIAL_NO


Output Streams:
--------------
   2) To Operator #3

      Estimated number of rows: 0.04352
      Partition Map ID:  1
      Partitioning:      (MULT )
                   Multiple Partitions
      Number of columns: 5
      Subquery predicate ID: Not Applicable
```

```
                Column Names:
                ------------
                +Q3.FIRST_NAME(A)+Q3.DEPARTMENT_NO(A)
                +Q3.AGE(A)+Q3.$RID$+Q3.LAST_NAME

                Partition Column Names:
                ----------------------
                +1: Q3.SERIAL_NO


      5) TQ    : (Table Queue)
          Cumulative Total Cost: 42.2932
          Cumulative CPU Cost: 402811
          Cumulative I/O Cost: 1
          Cumulative Re-Total Cost: 11.6513
          Cumulative Re-CPU Cost: 261762
          Cumulative Re-I/O Cost: 0
          Cumulative First Row Cost: 36.1495
          Cumulative Comm Cost:4.03306
          Cumulative First Comm Cost:0
          Estimated Bufferpool Buffers: 1

          Arguments:
          ---------
          LISTENER: (Listener Table Queue type)
             TRUE
          TQMERGE : (Merging Table Queue flag)
             FALSE
          TQREAD  : (Table Queue Read type)
             READ AHEAD
          TQSEND  : (Table Queue Write type)
             BROADCAST
          UNIQUE  : (Uniqueness required flag)
             FALSE

          Input Streams:
          -------------
             5) From Operator #6

                Estimated number of rows: 2.4
                Partition Map ID:  1
                Partitioning:      (CORR )
                             Directed to Single Partition based on correlation
value
                Number of columns: 0
                Subquery predicate ID: 9

                Partition Column Names:
```

```
                ----------------------
                +1: Q3.BONUS


        Output Streams:
        --------------
           6) To Operator #3

                Estimated number of rows: 4.8
                Partition Map ID:  1
                Partitioning:      (MULT )
                             Multiple Partitions
                Number of columns: 0
                Subquery predicate ID: 9

                Partition Column Names:
                ----------------------
                +NONE


    6) TBSCAN: (Table Scan)
        Cumulative Total Cost: 37.3742
        Cumulative CPU Cost: 309355
        Cumulative I/O Cost: 1
        Cumulative Re-Total Cost: 10.4705
        Cumulative Re-CPU Cost: 261762
        Cumulative Re-I/O Cost: 0
        Cumulative First Row Cost: 31.2304
        Cumulative Comm Cost:0
        Cumulative First Comm Cost:0
        Estimated Bufferpool Buffers: 1

        Arguments:
        ---------
        MAXPAGES: (Maximum pages for prefetch)
           1
        PREFETCH: (Type of Prefetch)
           NONE
        ROWLOCK : (Row Lock intent)
           NEXT KEY SHARE
        SCANDIR : (Scan Direction)
           FORWARD
        TABLOCK : (Table Lock intent)
           INTENT SHARE

        Predicates:
        ----------
        10) Sargable Predicate
           Relational Operator: Equal (=)
```

```
          Subquery Input Required: No
          Filter Factor: 0.02

          Predicate Text:
          --------------
          (Q3.BONUS = Q1.C1)


      Input Streams:
      -------------
          4) From Object CALISTO.T1

             Estimated number of rows: 120
             Partition Map ID:  1
             Partitioning:      (CORR )
                         Directed to Single Partition based on correlation
value
             Number of columns: 2
             Subquery predicate ID: 9

             Column Names:
             ------------
             +Q1.$RID$+Q1.C1

             Partition Column Names:
             ----------------------
             +1: Q3.BONUS


      Output Streams:
      --------------
          5) To Operator #5

             Estimated number of rows: 2.4
             Partition Map ID:  1
             Partitioning:      (CORR )
                         Directed to Single Partition based on correlation
value
             Number of columns: 0
             Subquery predicate ID: 9

             Partition Column Names:
             ----------------------
             +1: Q3.BONUS


Objects Used in Access Plan:
---------------------------
```

```
Schema: CALISTO
Name: EMP_INDEX
Type: Index
      Time of creation: 2003-11-19-19.29.10.180726
      Last statistics update:
      Number of columns: 4
      Number of rows: 34
      Width of rows: -1
      Number of buffer pool pages: 1
      Distinct row values: No
      Tablespace name: USERSPACE1
      Tablespace overhead: 24.100000
      Tablespace transfer rate: 0.900000
      Source for statistics: Single Node
      Prefetch page count: 32
      Container extent page count: 32
      Index clustering statistic: 80.000000
      Index leaf pages: 1
      Index tree levels: 1
      Index full key cardinality: 25
      Index first key cardinality: 25
      Index first 2 keys cardinality: -1
      Index first 3 keys cardinality: -1
      Index first 4 keys cardinality: -1
      Index sequential pages: 1
      Index page density: -1
      Index avg sequential pages: -1
      Index avg gap between sequences:-1
      Index avg random pages: -1
      Fetch avg sequential pages: -1
      Fetch avg gap between sequences:-1
      Fetch avg random pages: -1
      Index RID count: 0
      Index deleted RID count: 0
      Index empty leaf pages: 0
      Base Table Schema: CALISTO
      Base Table Name: EMPLOYEE
      Columns in index:
         LAST_NAME
         FIRST_NAME
         DEPARTMENT_NO
         AGE

Schema: CALISTO
Name: EMPLOYEE
Type: Table
      Time of creation: 2003-11-19-18.39.22.321031
      Last statistics update:
      Number of columns: 8
```

```
            Number of rows: 34
            Width of rows: 85
            Number of buffer pool pages: 1
            Distinct row values: No
            Tablespace name: USERSPACE1
            Tablespace overhead: 24.100000
            Tablespace transfer rate: 0.900000
            Source for statistics: Single Node
            Prefetch page count: 32
            Container extent page count: 32
            Table overflow record count: 0
            Table Active Blocks: -1

      Schema: CALISTO
      Name: T1
      Type: Table
            Time of creation: 2003-11-19-19.42.06.484719
            Last statistics update:
            Number of columns: 3
            Number of rows: 120
            Width of rows: 17
            Number of buffer pool pages: 1
            Distinct row values: No
            Tablespace name: USERSPACE1
            Tablespace overhead: 24.100000
            Tablespace transfer rate: 0.900000
            Source for statistics: Single Node
            Prefetch page count: 32
            Container extent page count: 32
            Table overflow record count: 0
            Table Active Blocks: -1
```

Guiding the DB2 optimizer towards a superior access path

DB2 provides the following SQL options/constructs that guide the DB2 optimizer in selecting an access path most suited for the application:

1. **`OPTIMIZE FOR n ROWS`** clause

   This clause as shown in Example 3-7 influences query optimization based on the assumption that n rows will be retrieved. This clause also determines the number of rows that are blocked in the communication buffer.

Example 3-7 OPTIMIZE FOR n ROWS

```
SELECT projno,projname,repemp FROM project
     WHERE deptno='D11' OPTIMIZE FOR 10 ROWS
```

Row blocking is a technique that reduces database manager overhead by retrieving a block of rows in a single operation, as described in “Use blocking cursors” on page 183. These rows are stored in a cache, and each **FETCH** request in the application gets the next row from the cache. If you specify **OPTIMIZE FOR 10 ROWS**, a block of rows is returned to the client every ten rows.

Specify the **OPTIMIZE FOR n ROWS** clause in the **SELECT** statement when:

– The number of rows you want to retrieve is significantly less than the total number of rows that could be returned.

– You need to access the first n rows quickly and can wait for the rest of the rows.

> **Note:** The **OPTIMIZE FOR n ROWS** clause does not limit the number of rows that can be fetched or affect the result in any way other than performance. Using **OPTIMIZE FOR n ROWS** can improve the performance if no more than n rows are retrieved, but may degrade performance if more than n rows are retrieved.

2. **FETCH FIRST n ROWS ONLY** clause

   Specify the **FETCH FIRST n ROWS ONLY** clause as described in Example 3-8 if you do not want the application to retrieve more than n rows, regardless of how many rows there might be in the result set when this clause is not specified. This clause cannot be specified with the **FOR UPDATE** clause.

Example 3-8 FETCH FIRST n ROWS ONLY

```
SELECT projno,projname,repemp FROM project
    WHERE deptno='D11'
    FETCH FIRST 5 ROWS ONLY
```

   Example 3-8 returns not more than 5 rows.

   The **FETCH FIRST n ROWS ONLY** clause also determines the number of rows that are blocked in the communication buffer. If both the **FETCH FIRST n ROWS ONLY** and **OPTIMIZE FOR n ROWS** clause are specified, the lower of the two values is used to determine the communication buffer size.

3. **FOR FETCH ONLY** clause

   When no cursor based updates are planned, specify the **FOR FETCH ONLY** clause in the **SELECT** statement. It can improve performance by allowing the query to take advantage of row blocking. It can also improve data concurrency, since only S locks will be taken on the rows retrieved by a query with this clause specified.

Note: The `FOR READ ONLY` clause is synonymous with the `FOR FETCH ONLY` clause.

Avoid data type conversions

Data type conversions, especially numeric data type conversions, should be avoided whenever possible.

When two values are compared, it may be more efficient to use items that have the same data type. Example 3-9 demonstrates a join of two tables, TableA and TableB, on columns A1 of TableA and column B1 of TableB.

Example 3-9 Data type conversions example

```
SELECT * FROM TableA,TableB WHERE A1=B1
```

If columns A1 and B1 are the same data type, no data type conversion is required. But if they are not the same data type, a data type conversion occurs to compare values at run time and it might affect the performance. For example, if A1 is a decimal column and B1 is an integer column and each has a value ‘123’, data type conversion is needed, as TableA stores it as x‘123C’, whereas TableB stores it as x‘7B’.

Also, inaccuracies may result when data type conversions occur due to limited precision.

Minimize network overheads

Applications accessing DB2 over a network should adopt the following coding techniques to achieve superior performance:

1. Use blocking cursors
2. Use compound SQL
3. Use Deferred Prepare
4. Use stored procedures

Use blocking cursors

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

The cache is allocated when an application issues an `OPEN CURSOR` request and is deallocated when the cursor is closed. The size of the cache is determined by the following database configuration parameters:

- For local applications, the database configuration parameter **`aslheapsz`**.
- For remote applications, database configuration parameter **`rqrioblk`** on the client workstation; this cache is allocated on the database client.

The **`FETCH FIRST n ROWS ONLY`** clause and **`FOR FETCH ONLY`** clause described in “Guiding the DB2 optimizer towards a superior access path” on page 181 enable row blocking.

Use compound SQL

Compound SQL is a technique to build one executable block from several SQL statements. When compound SQL is being executed, each SQL statement in the block is executed individually, but the number of requests transmitting between client and server is reduced.

Compound SQL is supported in stored procedures and the following application development processes:

- Embedded static SQL
- DB2 Call Level Interface
- JDBC

Example 3-10 shows compound SQL executing two **`UPDATE`** statements and one **`INSERT`** statement.

Example 3-10 Compound SQL

```
EXEC SQL BEGIN COMPOUND ATOMIC STATIC
    UPDATE tablea SET cola = cola * :var1;
    UPDATE tableb SET colb = colb + :var2;
    INSERT INTO tablec (colc,cold,cole) VALUES (:i,:j,0);
  END COMPOUND;
```

Compound SQLs may be atomic or non-atomic; the type determines how the entire block is handled when one or more SQL statements in the block happen to end in error.

- *Atomic* specifies that if any SQL statement in the block ends in error, the entire block is considered to have ended in error, and any changes made to the database within the block should be rolled back.
- *Non-atomic* specifies that when all statements have completed the application receives a response. Even if one or more statements in the block end in error, the database manager attempts to execute all statements in the block. If the unit of work containing the compound SQL is rolled back, then all changes made to the database within the block will be rolled back.

Use Deferred Prepare

Dynamic SQL is prepared at run time, static SQL is prepared at precompile time. As well as requiring more processing, the preparation step may incur additional network traffic at run time. The additional network traffic can be avoided if the DB2 CLI application makes use of deferred prepare, which is the default behavior.

Deferred prepare is the name of the CLI feature that seeks to minimize communication with the server by sending both the prepare and execute requests for SQL statements in the same network flow. The default value for this property can be overridden using the CLI/ODBC configuration keyword **`DEFERREDPREPARE`**. This property can be set on a per-statement handle basis by calling SQLSetStmtAttr() to change the **`SQL_ATTR_DEFEFERRED_PREPARE`** statement attribute.

When deferred prepare is on, the prepare request is not sent to the server until the corresponding execute request is issued. The two requests are then combined into one command/reply flow (instead of two) to minimize network flow and to improve performance. Because of this behavior, any errors that would typically be generated by SQLPrepare() will appear at execute time, and SQLPrepare() will always return **`SQL_SUCCESS`**.

Deferred prepare is of greatest benefit when the application generates queries where the answer set is very small, and the overhead of separate requests and replies is not spread across multiple blocks of query data.

Note: Even if deferred prepare is enabled, operations that require a statement to be prepared prior to the operation's execution will force the prepare request to be sent to the server before the execute. Describe operations resulting from calls to SQLDescribeParam() or SQLDescribeCol() are examples of when deferred prepare will be overridden, because describe information is only available after the statement has been prepared.

Use stored procedures

Stored procedures are similar to applications running on the client in that they can contain SQL and business logic, but unlike client applications they are stored on the database server, and run as functions invoked through the DB2 engine by the client. Because they execute on the database server itself, and retrieve, manipulate and store data without crossing the network, they can reduce network traffic, and may improve performance. Stored procedures can also be used to centralize and standardize business rules across business applications. They facilitate business rule reuse by being centrally located in the database. Since stored procedures consume CPU cycles on the database server, they may not be appropriate if CPU cycles are scarce on the database server.

Best practices

We recommend the following best practices for achieving superior performance with stored procedures:

1. Consider the use of stored procedures when an independent block of SQL statements and business logic retrieves or manipulates more than a few rows of data, and/or when a block of SQL or other business logic is applicable to multiple business applications.
2. Do not use stored procedures for trivial client or application requests across the network, since there is an overhead associated with invoking stored procedures.
3. For OLTP, a stored procedure should generally be used when a transaction has more than 4 or 5 SQL statements.
4. Stored procedures written in C using static SQL, or using SQL in PSM, typically give the best performance since they avoid the overhead of preparing SQL statements on each stored procedure invocation.

Minimize concurrency problems

Applications should be written to reduce the cost of locking and minimize locking contention when accessing shared data, as discussed in "Concurrency" on page 186.

3.3.7 Concurrency

Concurrency is critical to the performance and throughput of multi-user environments accessing shared data. Throughput and concurrency depend upon database configuration parameters and application design.

Database configuration parameters **LOCKLIST**, **MAXLOCKS** and **DLCHKTIME** impact concurrency and throughput, and are described in detail in 3.4.6, "Locking considerations" on page 260; they are only discussed in passing here. In this section, we focus mainly on application design considerations.

This section is organized as follows:

- Brief overview of DB2 locking
- Performance considerations
- Application locking considerations

Brief overview of DB2 locking

DB2 has a sophisticated locking implementation that enables a wide range of capabilities, from coarse table level locks to fine granularity row level locks, all under user control.

DB2 locks different database objects including databases, table spaces, tables, blocks (MDCs only) and rows.

In the following sections, we briefly review the following:

- Hierarchy of locks
- Lock type compatibility
- Lock conversion
- Isolation level
- Lock escalation
- Deadlocks

Hierarchy of locks

The hierarchy of DB2 locks is shown in Figure 3-13.

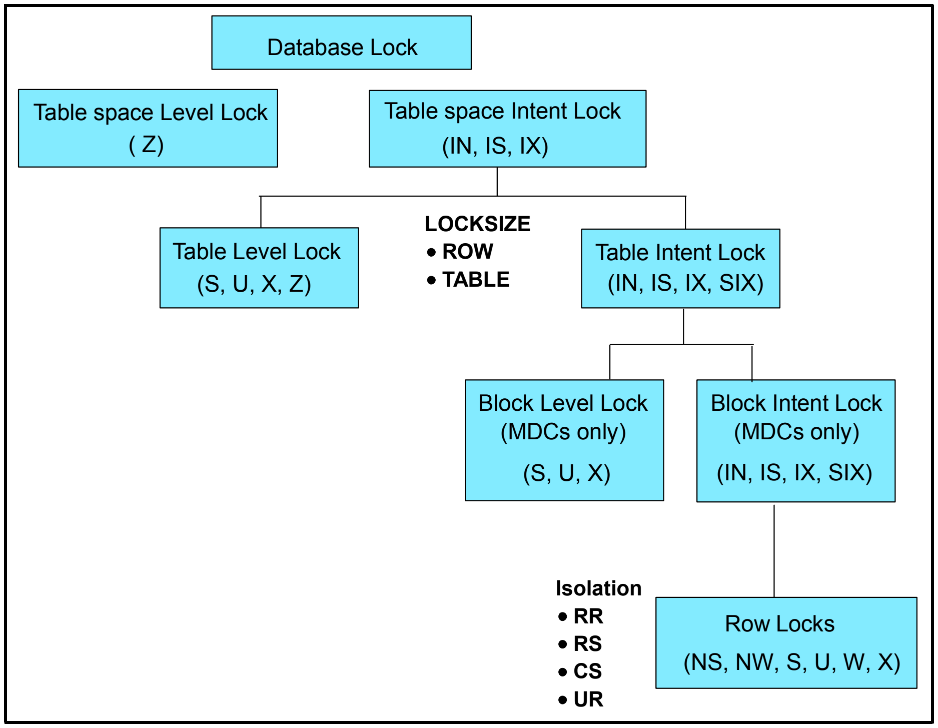

Database Lock
Table space Level Lock (Z)
Table space Intent Lock (IN, IS, IX)
Table Level Lock (S, U, X, Z)
LOCKSIZE
• ROW
• TABLE
Table Intent Lock (IN, IS, IX, SIX)
Block Level Lock (MDCs only) (S, U, X)
Block Intent Lock (MDCs only) (IN, IS, IX, SIX)
Isolation
• RR
• RS
• CS
• UR
Row Locks (NS, NW, S, U, W, X)

Figure 3-13 Lock hierarchy

A listing and brief description of the various locks is provided in Table 3-7 on page 188.

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

Table 3-7 Lock modes shown in order of increasing control over resources

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 cannot update, the locked data.

Lock mode	Applicable object type	Description
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 cannot 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.

Lock mode	Applicable object type	Description
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.
Z (Superxclusive)	Table space, 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.

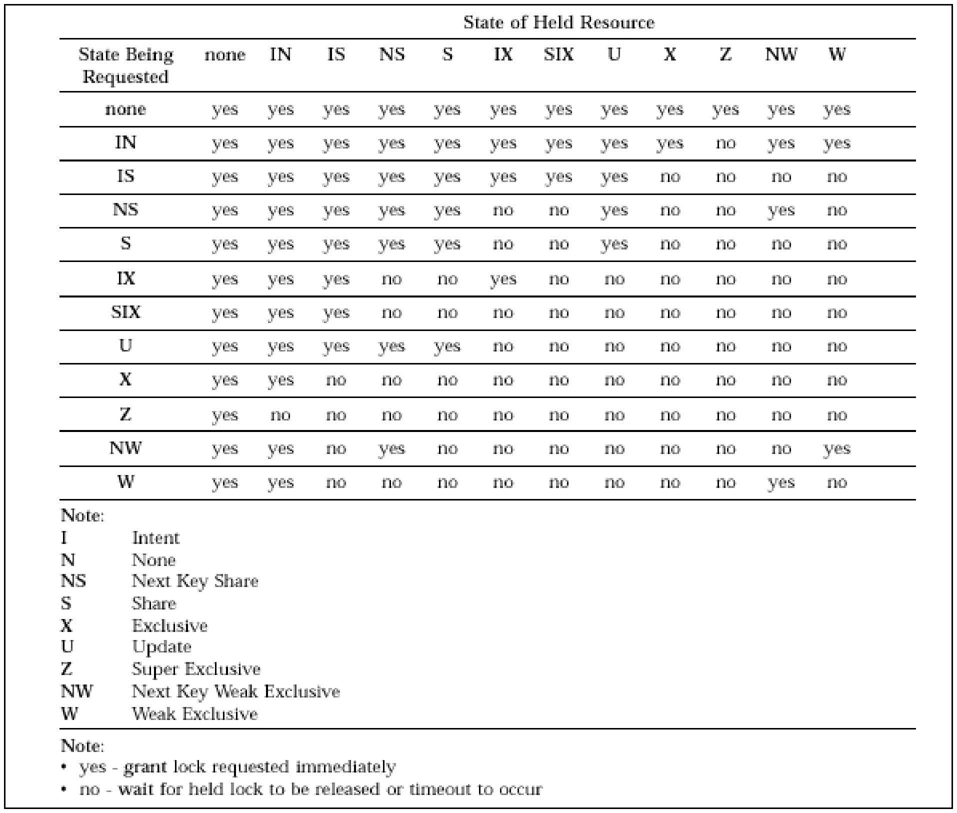
Important: When transactions cause changes to type-1 indexes, some next key locking occurs. For type-2 indexes however, there is minimal next key locking.

Type-2 indexes offer significant performance and availability advantages over type-1 indexes, and you are strongly encouraged to migrate existing type-1 indexes to type-2 indexes.

For details on type-2 indexes, refer to *IBM DB2 UDB Administration Guide: Performance*, SC09-4821.

Lock type compatibility

Figure 3-14 describes the compatibility matrix of the various lock modes. A `no` indicates that the requestor must wait until all incompatible locks are released by other processes. A timeout can occur waiting for a lock. A `yes` indicates that the lock is granted, unless someone else is waiting for the resource.



	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

Figure 3-14 Lock type compatibility

Lock conversion

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

Isolation level

An isolation level determines how data is locked or isolated from other processes, while the data is being accessed. The isolation level chosen is effective for the duration of the unit of work.

DB2 supports the following isolation levels:

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

RR is the most restrictive, while UR is the least restrictive. Table 3-8 summarizes the different isolation levels.

Table 3-8 Summary of different isolation levels

Isolation Level	**Access to Un-committed data**	**Non-repeatable reads**	**Phantom Read Phenomenon**
Repeatable Read	Not possible	Not possible	Not possible
Read Stability	Not possible	Not possible	Possible
Cursor Stability	Not possible	Possible	Possible
Uncommitted Read	Possible	Possible	Possible

Table 3-9 provides a simple, heuristic way to help you choose an initial isolation level for your applications. Consider this table a starting point; however, a thorough understanding of the semantics of the various isolation levels and their impact on concurrency might make another isolation level more appropriate.

Table 3-9 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.

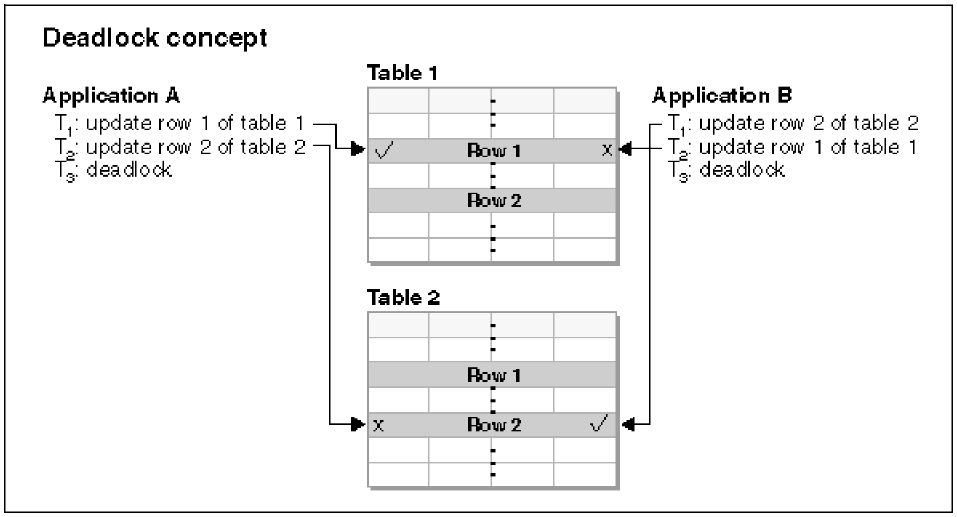
Lock escalation

Lock escalation is an internal mechanism that is invoked by the DB2 lock manager to reduce the number of locks held. Escalation occurs from row locks to a table lock when the number of locks held exceed the thresholds defined by the database configuration parameters **`LOCKLIST`** and **`MAXLOCKS`**.

Lock escalation can significantly impact concurrency and degrade response times of concurrent applications.

Deadlocks

Deadlocks occur when more than two or more applications wait on one another for resources that are held by the other. None of the applications can proceed until at least one of the waiting applications is forced to relinquish its lock. A process is required to break these deadlock situations. Figure 3-15 describes the deadlock concept.



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

Figure 3-15 Deadlock scenario

A user may control the granularity of locking via the **`LOCKSIZE`** parameter of the **`CREATE/ALTER TABLE`** statement, and the isolation levels specified at bind (**`ISOLATION`** parameter) or the **`WITH`** option on the **`SELECT`** statement.

The default is row locking for the table, and cursor stability (CS) for the isolation level.

Important: Locks are usually taken implicitly on behalf of an application during the execution of a query. Understanding the number and kinds of locks obtained on DB2 objects is helpful to effective tuning for maximum concurrency.

For details on lock modes, isolation levels, lock isolations and deadlocks, refer to *DB2 UDB Administration Guide: Performance*, SC09-4821.

Performance considerations

The trade-off of locking is concurrency versus lock processing cost.

- Greatest concurrency is achieved when each application takes the smallest possible lock on the least number of rows and holds it for the shortest possible duration; this means taking row locks, `UR` or `CS` isolation levels and committing frequently. However, this tends to incur significant CPU processing cost as well as memory utilization for locks.
- Acquiring a table lock incurs the least processing cost because only one lock is held and memory utilization is only a few bytes. However, concurrency can be significantly impacted if concurrent access to the table is desired by readers and updaters.

  Note: The exception is a read only environment such as in data warehousing where table locks combine the advantages of high concurrency and low lock processing cost.

Application locking considerations

Applications should be designed to balance concurrency with low processing cost.

In most cases, applications try to provide maximum concurrency while minimizing lock waits, lock timeouts, deadlocks and lock processing overhead. As mentioned earlier, this is achieved by holding the smallest possible lock on the least number of rows and holding it for the shortest possible duration.

Best practices

We recommend the following best practices for achieving maximum concurrency with minimum lock contention:

1. Let the `LOCKSIZE` parameter in the `CREATE TABLE` default to row locking so that the smallest lock size is requested.
2. Commit frequently to reduce the duration a lock is held. For batch processing, in an OLTP environment, a reasonable starting point for commits is after every

500 to 1000 updates corresponding to transactional boundaries, which should then be tuned to address the specific requirements of your environment. Committing too frequently can increase concurrency, but negatively impact performance because of the overheads of flushing log buffers and lock release costs.

3. Avoid `LOCK TABLE` statement with `EXCLUSIVE` mode

   This statement promotes an existing lock on a table to a most restrictive state, which can have a very detrimental impact on concurrency and overall system throughput.

4. Specify the `FOR FETCH ONLY` clause in the `SELECT` statement as described in "FOR FETCH ONLY clause" on page 182. This results in the acquiring of less restrictive S share locks conducive to greater concurrency.

5. Specify the `FOR UPDATE` clause in a cursor definition to minimize the cost of lock conversions as well as potential deadlocks.

   A cursor based update should have the `FOR UPDATE` clause in the `SELECT` statement of the cursor definition. This results in the database manager choosing U (update) locks instead of S (shared) locks — this saves the cost of performing lock conversions from S locks to U locks when the succeeding `UPDATE` statement is processed.

   The other benefit to specifying the `FOR UPDATE` clause is that it can decrease the possibility of deadlock as follows:

   Assume two applications each using cursor stability isolation are trying to fetch the same row and update it simultaneously using cursors without the `FOR UPDATE` clause in the following order as shown in Figure 3-16:

   – Step 1: Application1 fetches the row with an S lock on it.
   – Step 2: Application2 fetches the same row with an S lock on it as well since there is no lock conflict with shared locks.
   – Step 3: Application1 attempts to update the row; it needs to convert its S lock to a U lock, but has to wait till Application2 releases its S lock.
   – Step 4: Application2 attempts to update the row; it needs to convert its S lock to a U lock, but has to wait till Application1 releases its S lock.

   Both applications wait on each other to release its lock, resulting in a deadlock!

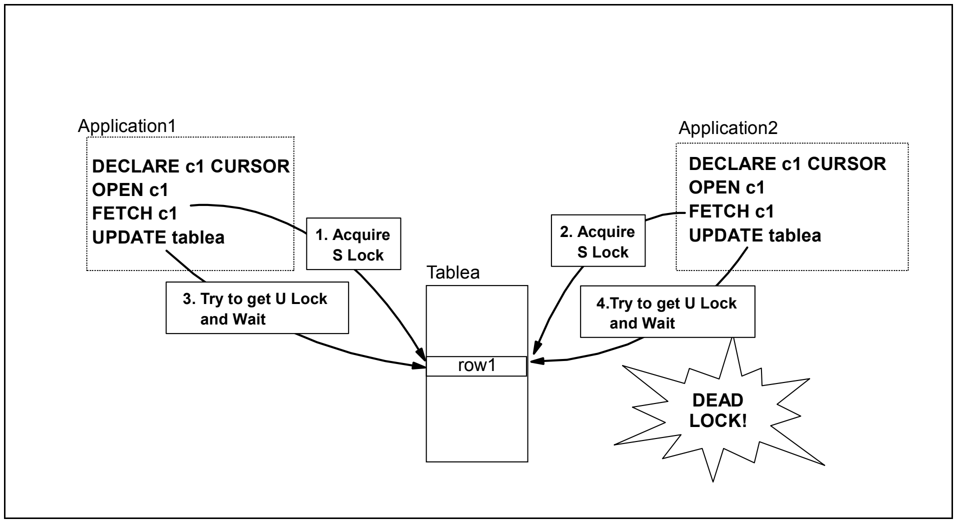


Figure 3-16 Cursor-based deadlocks without FOR UPDATE clause

With the **`FOR UPDATE`** clause in the **`DECLARE CURSOR`** statement as shown in Example 3-11, Application1 fetches the row with a U lock in step 1, but when Application 2 tries to fetch the same row in step 2 with a U lock, it has to wait until Application1 releases its U lock. Deadlocks are thus avoided, but lock waits occur and a timeout may even happen, depending upon how long Application2 waits for the lock.

Example 3-11 Cursor definition with FOR UPDATE clause

```
EXEC SQL DECLARE c1 CURSOR FOR select * from employee
   FOR UPDATE OF job;
EXEC SQL OPEN c1;
EXEC SQL FETCH c1 INTO...;
if ( strcmp (change,"YES") == 0)
EXEC SQL UPDATE employee SET job=:newjob
  WHERE CURRENT OF c1;
EXEC SQL CLOSEc1;
```

CLI programs can set **`SQL_MODE_READ_WRITE`** to the DB2 CLI connection attribute **`SQL_ATTR_ACCESS_MODE`** using the SQLSetConnectAttr() function to achieve the same results.

For further details, refer to *DB2 UDB Call Level Interface Guide and Reference, Volume 1*, SC09-4849, and *DB2 UDB Call Level Interface Guide and Reference, Volume 2*, SC09-4850.

6. Perform SQL **`INSERT, UPDATE`** and **`DELETE`** at the end of a unit of work, if possible. This narrows the window of restrictive X locks, which enhances concurrency.

7. Consider `LOCKSIZE TABLE` under the following circumstances since it incurs the least lock processing cost, because no row locks are taken and only a single lock is acquired with this option:
   - For read only tables in environments such as data warehousing. Concurrency is not an issue since all applications request share locks that are compatible with each other.
   - For tables which are only updated by a single application in a "batch window". Concurrency is not an issue, even though an X lock would be taken on the table since there is no concurrent access to the table during batch processing.
8. Choose the minimally restrictive isolation level required for the application. The order from least restrictive to most restrictive level is `UR, CS, RS` and `RR`. This results in row locks being held for the smallest duration.
   - Use the `UR` isolation level for queries against read-only tables, or when access to uncommitted data is acceptable. This isolation level provides maximum concurrency.
   - Use the `CS` isolation level for maximum concurrency when access to only committed data is acceptable.
   - Use the `RS` isolation level when the application requires qualified rows to remain stable for the duration of the unit of work.
   - Use the `RR` isolation level when the application requires the result set to be consistent for the duration of the unit of work. This isolation level provides the least concurrency.
9. Release read locks using the `WITH RELEASE` option of the `CLOSE CURSOR` statement if acceptable to the application; this reduces the duration the locks are held and therefore enhances concurrency.

   Note that this only applies to cursors with the `RR` and `RS` isolation level. The normal operation without the `WITH RELEASE` option is that locks get released at the end of the unit of work.
10. Access objects in applications in a sequence that supports a high degree of concurrency with minimal deadlocks. For example:
    - Assume application A first accesses table T1, then updates table T1, followed by access to table T2 and update of table T2.
    - Assume application B first accesses table T2, then updates table T2, followed by access to table T1 and update of table T1.

    This sequence of processing can result in deadlocks if both applications need access to the individual rows locked by the other. By ensuring that both applications access and update tables T1 and T2 in the same sequence, deadlocks can be minimized. However, lock waits and timeouts may still occur.

11. Tune the database configuration parameters **LOCKLIST, MAXLOCKS** and **DLCHKTIME** to minimize lock escalations and optimize deadlock detection cycles as described in 3.4.6, "Locking considerations" on page 260.
12. OLTP environments are high throughput and transactional in nature, and should therefore use row level locking with **CS** isolation level unless there is a specific reason for a more restrictive lock. They should also commit more frequently to reduce the duration a lock is held.
13. BI environments are low throughput and predominantly read only in nature, and either table level locks or row locks with **CS** or **UR** isolation level would be appropriate.

3.4 System environment considerations

While application design is critical, there are a number of system considerations that can significantly impact overall system performance. System tuning is typically performed by DBAs in conjunction with operating system administrators.

The following system performance-related considerations are discussed here:

- I/O placement considerations
- Log considerations
- Monitor switch settings
- Connection considerations
- Buffer pool considerations
- Locking considerations
- Package cache considerations
- Catalog cache considerations
- Sort considerations
- Other memory considerations
- Miscellaneous considerations

3.4.1 I/O placement considerations

Proper disk subsystem placement of critical DB2 objects is essential to reducing I/O times and ensuring high availability of these objects.

Figure 3-17 on page 199 describes the default directory structure for a database. Figure 3-18 on page 200 highlights the main DB2 objects.

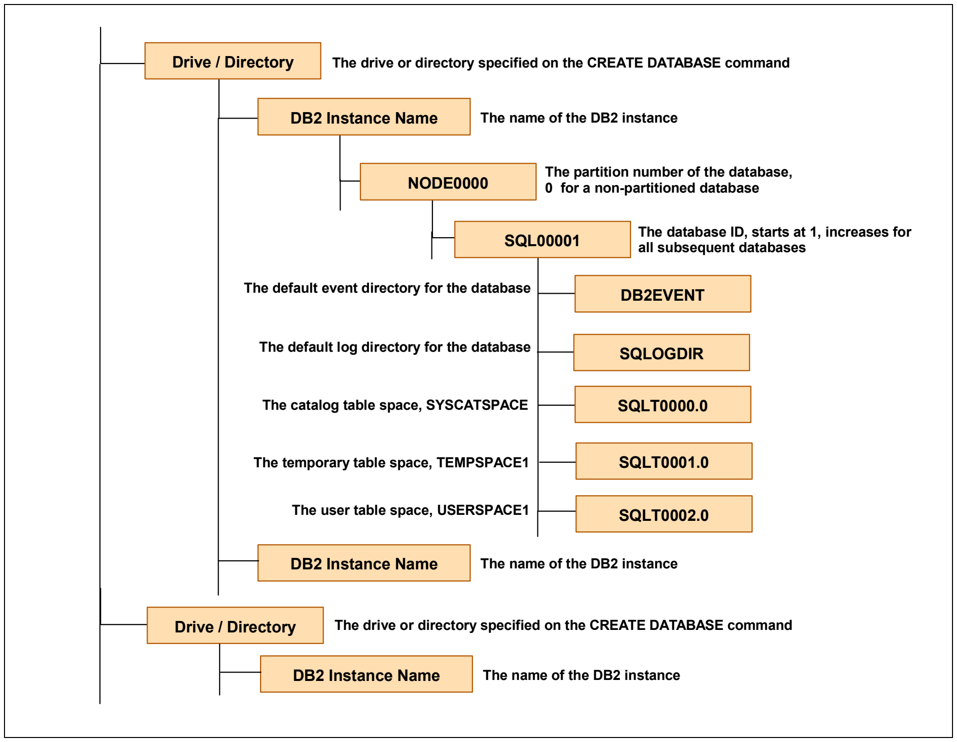

Drive / Directory
The drive or directory specified on the CREATE DATABASE command
DB2 Instance Name
The name of the DB2 instance
NODE0000
The partition number of the database, 0 for a non-partitioned database
SQL00001
The database ID, starts at 1, increases for all subsequent databases
The default event directory for the database
DB2EVENT
The default log directory for the database
SQLOGDIR
The catalog table space, SYSCATSPACE
SQLT0000.0
The temporary table space, TEMPSPACE1
SQLT0001.0
The user table space, USERSPACE1
SQLT0002.0
DB2 Instance Name
The name of the DB2 instance
Drive / Directory
The drive or directory specified on the CREATE DATABASE command
DB2 Instance Name
The name of the DB2 instance

Figure 3-17 Default database directory structure

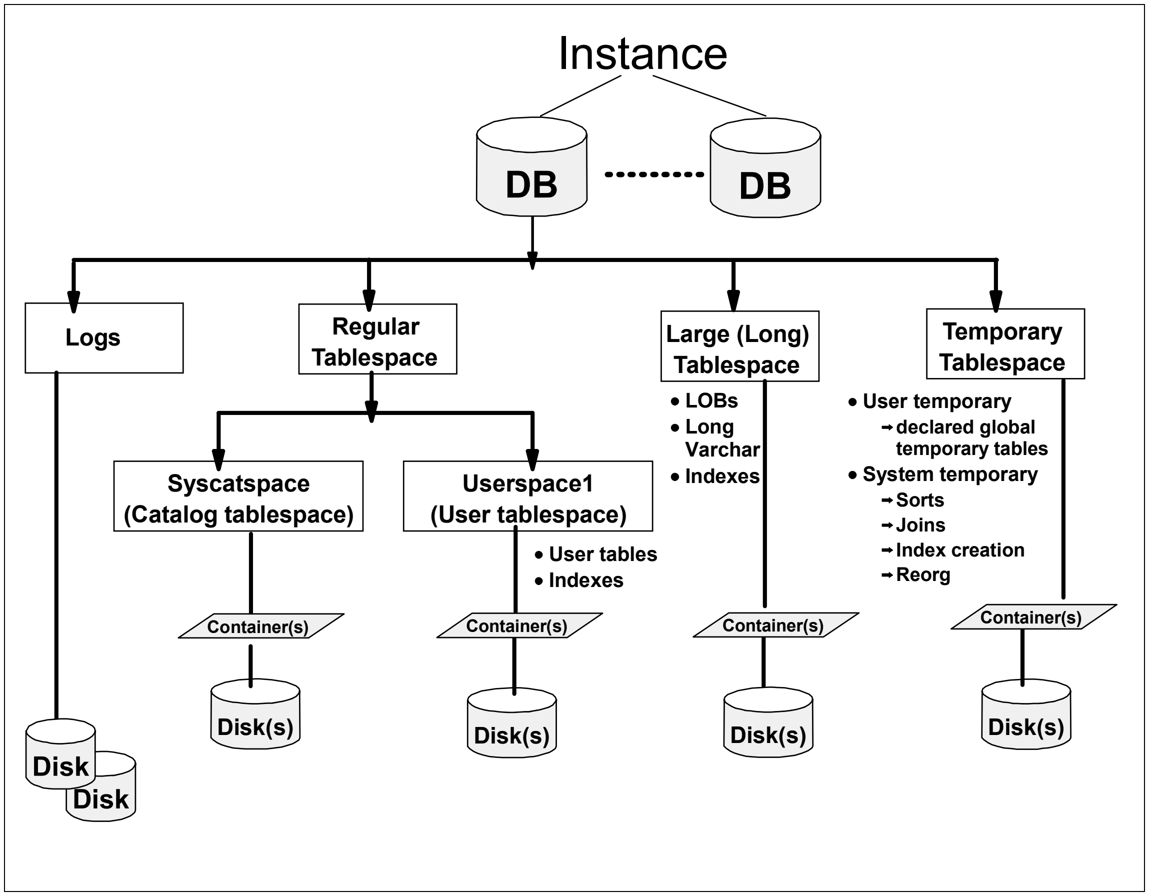


Figure 3-18 DB2 objects placement

Figure 3-18 highlights the four categories of DB2 objects as follows:

- DB2 logs.
- Regular table spaces, which include:
  - User data and indexes in the default **USERSPACE1** table space or in one or more user-defined table spaces.
  - System catalogs that store DB2 metadata in the **SYSCATSPACE** catalog table space.
- Large table space (previously called Long table space) that stores LOBs, Long Varchar columns and indexes.
- Temporary table space that consists of two types of table spaces:
  - User temporary table spaces that store declared global temporary tables.

– System temporary table spaces that are used as spill areas for tasks such as sorts, joins, index creation and reorgs.

Performance considerations

A well-balanced I/O subsystem is critical to overall system performance. Placing DB2 objects with contending access patterns on the same disk subsystem can negatively impact overall system performance and should be avoided if possible.

Best practices

We recommend the following best practices for DB2 object placement to achieve superior performance and availability:

1. A general rule of thumb for the amount of disk space required for a database is four times the estimated size of raw data.
2. The general rule of thumb for a database is a minimum of 6 to 10 disks per CPU (excluding operating system and other component requirements). If there are fewer than 6 disks per CPU, then spread all the table spaces across all the available physical disks. If there are more than 6 to 10 disks per CPU, then assign table spaces to different disks as per table activity/priority.

   Note: Spreading “everything over everything” (all table spaces on all devices) is easy and generally produces good results.

   In scenarios where you need to prioritize different applications, you may isolate those applications’ high priority DB2 objects to different devices in order to achieve superior performance for those applications. However, this could increase the risk of creating a poorly performing configuration.
3. DB2 logs are critical DB2 objects that need to be protected. Ideally (assuming sufficient number of disks) logs should be placed on separate disks and disk controllers totally isolated from all the other DB2 objects, as this will allow for efficient logging activity with a minimum of overhead, such as waiting for I/O. DB2 logs should also be duplicated using either disk mirroring or dual logging.

   Ensure that the file system for logs is created using the “large file enabled” option as described in “Filesystem recommendations” on page 359.
4. The system catalog table space should be isolated on separate disks; the default SMS table space should be adequate in most environments.
5. User table space contains user tables and indexes, and the assignment of these SMS or DMS table spaces to disks should be on the basis of their individual activity and priority characteristics. In general, the isolation scheme should be based on the table space activity/priority as shown in Table 3-10, assuming there are enough disks available.

Table 3-10 Table space activity/priority compatibility matrix

	High activity/priority	**Medium activity/priority**	**Low activity/priority**
High activity/priority	No	No	Yes
Medium activity/priority	No	Maybe	Yes
Low activity/priority	No	Yes	Yes

6. Large table spaces contain LOB and LONG VARCHAR column data. Reads and writes to this table space bypass the buffer pool, and therefore benefit by being defined as SMS table spaces or DMS table spaces that use file containers, since they both exploit the file system cache.

   In general, these table spaces should be spread across multiple disks and containers based on capacity and activity/priority considerations.

7. System temporary table spaces (for different page sizes) should be SMS table spaces and should also be assigned to separate disks if possible. A large buffer pool is highly recommended to avoid sort temporary tables spilling over to disk, especially in OLTP environments. Consider a minimum of 3 to 4 containers for a system temporary table space.

   – OLTP environments generally do not have many or large sorts, and a single system temporary table space should be adequate spread over multiple containers. The default 4 K page size is generally enough in these environments.

   – BI environments, on the other hand, tend to have many large sorts and may use multiple page sizes. It is desirable to have only one system temporary table space per page size. You may need to create table spaces with a page size larger than 4 K (and corresponding buffer pools), depending upon the row length and columns used in the tables in the database. A separate buffer pool may be appropriate for temporary table spaces in BI environments.

8. User temporary table spaces have considerations similar to user table spaces containing user data in DGTTs. A single SMS table space spread across multiple disks for each page size required is probably enough in most cases.

Performance monitoring metrics

To determine whether the I/O placement of DB2 objects are satisfactory requires analysis of the combined results of table space activity using DB2 commands, and device utilization using operating system commands.

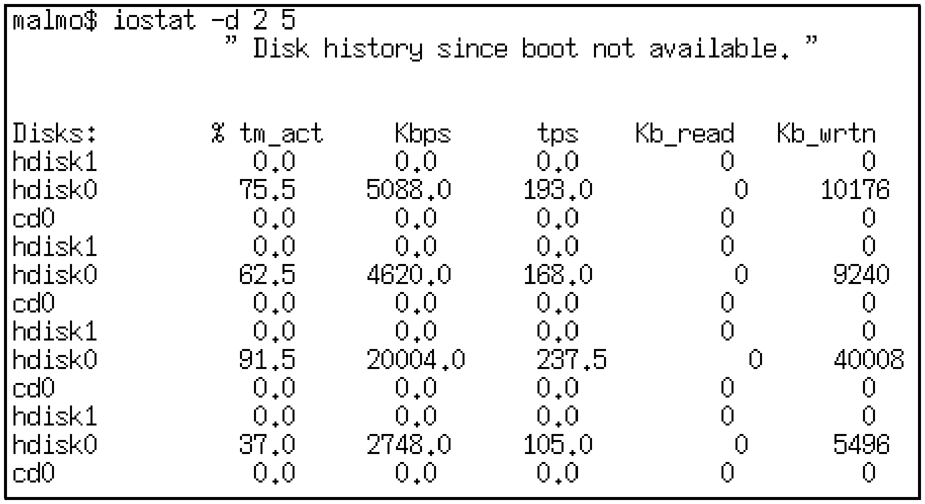
Important: This information needs to be collected over a reasonable time period in order to identify reliable trends for proper analysis.

The steps involved are:

1. Collect disk utilization statistics
2. Identify table space activity
3. Map file systems to underlying disks
4. Determine relationship between disk utilization and table spaces
5. Revisit I/O placement decisions, if necessary

Collect disk utilization statistics

Collect disk utilization statistics using operating system tools such as **`iostat`** in AIX as shown in Figure 3-19. The % tm_act column lists the percentage[5] of time the disk was busy representing bandwidth utilization, while the tps column shows the number of transfers per second to each disk.



```
malmo$ iostat -d 2 5
               " Disk history since boot not available. "


Disks:        % tm_act     Kbps      tps    Kb_read   Kb_wrtn
hdisk1           0.0       0.0       0.0         0         0
hdisk0          75.5    5088.0     193.0          0     10176
cd0              0.0       0.0       0.0         0         0
hdisk1           0.0       0.0       0.0         0         0
hdisk0          62.5    4620.0     168.0          0      9240
cd0              0.0       0.0       0.0         0         0
hdisk1           0.0       0.0       0.0         0         0
hdisk0          91.5   20004.0      237.5          0     40008
cd0              0.0       0.0       0.0         0         0
hdisk1           0.0       0.0       0.0         0         0
hdisk0          37.0    2748.0     105.0          0      5496
cd0              0.0       0.0       0.0         0         0
```

Figure 3-19 AIX iostat command results

For Windows, refer to 5.3.4, "Monitoring and problem determination tools" on page 387.

Identify table space activity

Identify table space activity of all table spaces, as shown in Example 3-12, using the following command:

```
db2 get snapshot for tablespaces on <db_alias>
```

This provides details of table space activity and container information, as well as buffer pool activity.

[5] A value greater than 40% typically indicates a problem.

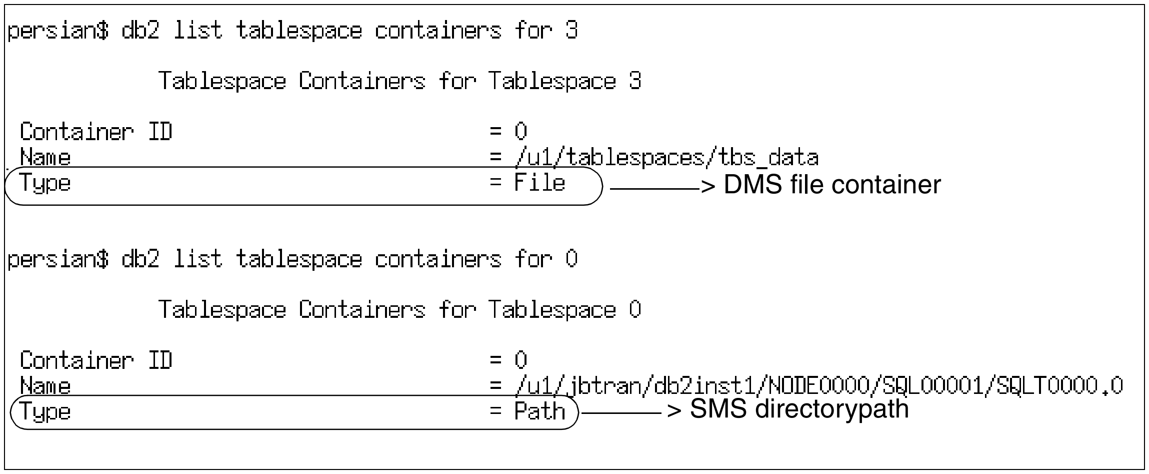
Example 3-12 Table space snapshot

```
db2 => get snapshot for tablespaces on sample

             Tablespace Snapshot

First database connect timestamp           = 10-15-2003 17:30:31.379938
Last reset timestamp                       =
Snapshot timestamp                         = 10-15-2003 17:31:19.521976
Database name                              = SAMPLE
Database path                              = C:\DB2\NODE0000\SQL00002\
Input database alias                       = SAMPLE
Number of accessed tablespaces             = 3

..........

Tablespace name                            = USERSPACE1
  Tablespace ID                            = 2
  Tablespace Type                          = System managed space
  Tablespace Content Type                  = Any data
  Tablespace Page size (bytes)             = 4096
  Tablespace Extent size (pages)           = 32
  Tablespace Prefetch size (pages)         = 16
  Buffer pool ID currently in use          = 1
  Buffer pool ID next startup              = 1
  Tablespace State                         = 0x'00000000'
   Detailed explanation:
     Normal
  Total number of pages                    = 694
  Number of usable pages                   = 694
  Number of used pages                     = 694
  Minimum Recovery Time                    =
  Number of quiescers                      = 0
  Number of containers                     = 1

  Container Name                           = C:\DB2\NODE0000\SQL00002\SQLT0002.0

      Container ID                         = 0
      Container Type                       = Path
      Total Pages in Container             = 694
      Usable Pages in Container            = 694
      Stripe Set                           = 0
      Container is accessible              = Yes

  Buffer pool data logical reads           = 3
  Buffer pool data physical reads          = 1
  Asynchronous pool data page reads        = 0
  Buffer pool data writes                  = 0
  Asynchronous pool data page writes       = 0
```

```
Buffer pool index logical reads          = 0
Buffer pool index physical reads         = 0
Asynchronous pool index page reads       = 0
Buffer pool index writes                 = 0
Asynchronous pool index page writes      = 0
Total buffer pool read time (ms)         = 16
Total buffer pool write time (ms)        = 0
Total elapsed asynchronous read time     = 0
Total elapsed asynchronous write time    = 0
Asynchronous read requests               = 0
Direct reads                             = 0
Direct writes                            = 0
Direct read requests                     = 0
Direct write requests                    = 0
Direct reads elapsed time (ms)           = 0
Direct write elapsed time (ms)           = 0
Number of files closed                   = 0
Data pages copied to extended storage    = 0
Index pages copied to extended storage   = 0
Data pages copied from extended storage  = 0
Index pages copied from extended storage = 0
```

Note: For SMS table spaces, the container name displayed is a directory (C:\DB2\NODE0000\SQL00002\SQLT0002.0 in Example 3-12). For DMS table spaces, the name displayed is that of a file or a raw device, for example /u1/tablespaces/tbs_data.

The container name is available from the results of this command. It may also be obtained by issuing **`db2 list tablespace containers for <n>`** as shown in Figure 3-20 on page 206, where n is the tablespace id.



```
persian$ db2 list tablespace containers for 3

            Tablespace Containers for Tablespace 3

 Container ID                         = 0
 Name                                 = /u1/tablespaces/tbs_data
 Type                                 = File


persian$ db2 list tablespace containers for 0

            Tablespace Containers for Tablespace 0

 Container ID                         = 0
 Name                                 = /u1/jbtran/db2inst1/NODE0000/SQL00001/SQLT0000.0
 Type                                 = Path
```

Figure 3-20 List tablespace containers output

Map file systems to underlying disks

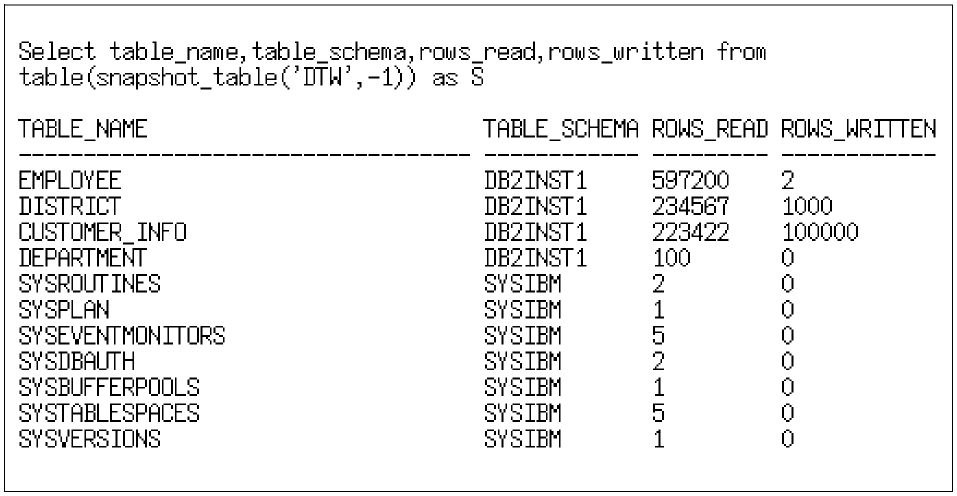
Determine the underlying physical disks of these containers for AIX using the method described in “Mapping filesystems to physical disks” on page 364.

For Windows, use the method described in “Mapping filesystems to physical disks” on page 388.

Determine relationship between disk utilization and table spaces

After collecting the disk utilization statistics, identifying the table space activity, and mapping the file systems to underlying disks, you can determine the relationship between high utilization disks, the table spaces located on each of the physical disks, and the activity level of individual table spaces on each physical disk.

Note: You can drill down to activity against specific tables in a multi-table by identifying the tables in the table space of interest, and then taking table snapshot information by using the SQL snapshot feature as shown in Figure 3-21 on page 207. This provides details of the number of rows read and written, which is the measure of activity against the respective tables.



```
Select table_name,table_schema,rows_read,rows_written from
table(snapshot_table('DTW',-1)) as S

TABLE_NAME                          TABLE_SCHEMA ROWS_READ ROWS_WRITTEN
----------------------------------- ------------ --------- ------------
EMPLOYEE                            DB2INST1     597200    2
DISTRICT                            DB2INST1     234567    1000
CUSTOMER_INFO                       DB2INST1     223422    100000
DEPARTMENT                          DB2INST1     100       0
SYSROUTINES                         SYSIBM       2         0
SYSPLAN                             SYSIBM       1         0
SYSEVENTMONITORS                    SYSIBM       5         0
SYSDBAUTH                           SYSIBM       2         0
SYSBUFFERPOOLS                      SYSIBM       1         0
SYSTABLESPACES                      SYSIBM       5         0
SYSVERSIONS                         SYSIBM       1         0
```

Figure 3-21 Output of table snapshot using SQL snapshot function

Revisit I/O placement decisions if necessary

The mapping generated in the previous step can be used to revisit I/O placement decisions to minimize I/O contention for superior performance.

The disk utilization information collected from sources such as health indicators **`Table Space Utilization`** (for DMS) and **`Table Space Container Utilization`** (for SMS) of the Health Center, as described in "Heath Monitor and Health Center" on page 100, can be used to track space utilization in order to allocate additional containers if required.

3.4.2 Log considerations

Logging is critical for maintaining the integrity, recoverability, and availability of the database environment. DB2 provides a number of logging options for managing the recoverability of the database environment; these options need to be configured carefully in order to achieve optimal logging performance.

In the following subsections, we provide an overview of DB2 logging and discuss the key decisions to be made for achieving superior logging performance.

- Logging overview
- Performance drivers

Logging overview

All databases have logs associated with them. These logs keep records of database changes and system activity. If a database needs to be restored to a point beyond the last full, offline backup, logs are required to roll the data forward to the point of failure. Logs are also required to restart the system after a crash such as a power failure; this is called *crash recovery*.

The following are some of the main characteristics of logging:

1. DB2 logs can be written to file systems or raw logical volumes. Using raw logical volumes for logging provides slightly improved performance compared to using UNIX directories.
2. DB2 implements dual logging when the **`mirrorlogpath`** database configuration parameter is configured with a path name.

   When dual logging is enabled, the log files are written sequentially. The log buffer is first flushed to the first log, and a write is initiated to the second log on completion of the write to the first log. This may have a slight impact on performance.

   Note: Disk mirroring technology may also be used to achieve higher log availability.
3. DB2 supports two types of logging, archive logging and circular logging.
   a. Archive logging

      When archiving is enabled, a log file is archived when it fills with log records. New log files are made available for log records. Archive logging is used specifically for rollforward recovery. Archive logging is enabled when the database configuration parameter **`logretain`** is set to **`recovery`** and/or the **`userexit`** database configuration parameter is set to **`yes`**.

      Note: Sample `USEREXIT` programs are present under <instance-home-directory>/sqllib/samples/c for AIX and <DB2 install directory>\IBM\SQLLIB\samples\c in Windows.

   b. Circular logging (the default)

      As the name suggests, circular logging uses a “ring” of online logs to provide recovery from transaction failures and system crashes. The logs are used and retained only to the point of ensuring the integrity of current transactions.

      The database configuration parameters **`logretain`** and **`userexit`** are both set to **`no`**. With this type of logging, only full, offline backups of the database are allowed. The database must be offline (inaccessible to users) when a full backup is taken.

      Circular logging does not allow you to roll a database forward through transactions performed after the last full backup operation. All changes occurring since the last backup operation are lost.

4. DB2 supports the concept of primary and secondary logs. One or more log files (**logprimary** database configuration parameter) may be defined to store log records. When they become full, spillover occurs to secondary logs (**logsecond** database configuration parameter). The primary log files establish a fixed amount of storage allocated to the recovery log files, and the **logprimary** parameter specifies the number of primary log files to be *preallocated*.

   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:

   – With circular logging, a primary log file can be reused when the changes recorded in it have been committed. If the size of the log (**logfilsiz** database configuration parameter) 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. 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 logs are dynamically deallocated when they are no longer needed by the database manager.

   – With archive logging, when a primary log file is full, the log is archived and a new primary log file is allocated.

   When the log disk is full, the action taken depends upon the setting of the **blk_log_dsk_full** database configuration parameter.

   – If set to **NO** (default), then a transaction that receives a log disk full error (**SQL0964N 'Log full'**) fails and is rolled back.

   – If set to **YES**, applications will hang when DB2 encounters a log disk full error; this allows the DBA to resolve the error and enables the applications to complete successfully.

     The log disk full situation can be resolved by moving old log files to another file system or by enlarging the file system, so that hanging applications can complete.

     Users will not be able to commit their transactions until the new log file is successfully created.

   DB2 attempts 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.

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.

5. DB2 defines the concept of active logs and archive logs

   – An active log is primarily meant to support transaction recovery and crash recovery; these logs are written with both archive and circular logging.

     These logs are currently in use by transactions. They are used during crash recovery. They are used to apply or undo the appropriate transactions to change the database to a consistent and usable stage.

     These logs are located in the directory listed in the informational database configuration parameter **`logpath`**.

     > **Note:** The maximum total active log space is 256 GB, unless infinite log space is defined by setting the **`logsecond`** database configuration parameter set to -1.
     >
     > When infinite log space is enabled, it allows transaction recovery to occur from archive logs as well, since it places no limit on the size or number of in-flight transactions running in the database. Active logs can be archived when infinite log space is enabled.
     >
     > Infinite log space requires archive logging; **`logretain`** must be set to **`recovery`** and/or **`userexit`** database configuration parameter must be set to **`yes`**.
     >
     > The default is infinite active log space is disabled.

   – An archive log is primarily for rollforward recovery, and is created when the user exit creates it or it is done manually. Archived logs are logs that were active but no longer required for crash recovery.

6. Log records are written to a log buffer (database configuration parameter **`logbufsz`**) and are flushed to disk by the logger process (db2loggr) under the following circumstances:

   – When transaction commits or a group of transactions commit as defined by the database configuration parameters **`mincommit`**.

– When one second has elapsed since the last log buffer flush.

– 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.

The default value is 8 x 4 K pages.

Note: The transaction issuing the commit has to wait until the log buffer has been flushed to disk. With dual logging, it has to wait until the write is completed to both logs.

7. Log records are read from the log during transaction rollback, crash recovery, and rollforward recovery.

Performance drivers

The following performance drivers need to be tuned for optimal performance of the logging system:

- Choice of the type of logging - archive or circular
- File system or raw logical volumes for the logs
- Single or dual logging
- **`logprimary, logsecond`** and **`logfilsiz`** db cfg parameters
- **`logbufsz`** db cfg parameter
- **`mincommit`** db cfg parameter
- **`blk_log_dsk_full`** db cfg parameter

In the following sections, each of these drivers is discussed in turn: their performance considerations, best practices, and performance monitoring metrics.

Choice of the type of logging - archive or circular

This is not directly related to performance as much as it is related to the need to support rollforward recovery.

- OLTP environments should choose archive logging, since they require rollforward recovery.
- BI environments should generally choose archival logging as well, unless it is a purely read-only environment, in which case circular logging may be appropriate.

The trend is towards updateable BI environments (for example, real time BI scenarios where transactions directly update data warehouses), and therefore archival logging is recommended.

File system or raw logical volumes for the logs

Raw logical volume I/Os generally outperform file system I/O, but there are other considerations as follows:

- The *advantages* of using raw logical volumes are:
  - The I/O path is shorter, since the operating system's file system is bypassed.
  - Raw device striping may provide faster I/O throughput.
  - There are no restrictions like those of size imposed by a file system.
  - A raw logical volume can span multiple disks.
- The *disadvantages* of using raw logical volumes are:
  - The device (logical volume) created for the logs must be dedicated to DB2.
  - There are limited tools and utilities available for operating on or backing up logs created on raw logical volumes, compared to tools for filesystem-based logs.
  - They can not take advantage of file system cache.

High throughput/performance OLTP environments should choose file system logging in general, unless the benefits of slightly superior performance of raw devices are critical to the application workload and outweigh their disadvantages.

BI environments and low-to-medium OLTP environments should typically not require the increased I/O performance benefits of raw devices; therefore, file system logging should be adequate.

Single or dual logging

While this is primarily an issue of availability, having dual logging requires twice the amount of disk space, and it also impacts response time, since the write to the second log is done serially. The additional response time overhead depends upon the average time to perform the log write I/O.

Mission-critical OLTP environments should use dual logging even though there may be a slight performance penalty that could affect high throughput/performance OLTP environments.

BI environments should probably find single logging sufficient, since such environments probably use circular logging and rollforward recovery is not required.

logprimary, logsecond, and logfilsiz db cfg parameters

The size and number of primary logs have an impact on disk space utilization since this space is preallocated. Secondary logs are allocated on an as needed basis.

From an active log space point of view:

- Overallocating `logprimary` can result in wasted space because this space is preallocated, unlike `logsecond`, which is allocated on demand.
- Having too small an active log space can result in log full conditions unless infinite log space is enable by setting `logsecond` to -1. While infinite log space prevents log full conditions, it has the potential to degrade performance of crash recovery and transaction rollback if the required log records need to be retrieved from the archived logs.

For a given amount of available disk capacity, the question that needs to be answered is whether it is more efficient to have a few primary logs with large file sizes, or many primary logs with smaller file sizes.

- With a larger log file size and a smaller number of primary logs:
  - The main *advantage* is of lower costs of switching logs as one becomes full.
  - The *disadvantages* are as follows:
    - Unacceptable data loss in log shipping to remote locations in support of disaster recovery scenarios.
    - With archive logging, if the log cannot be archived in time, then DB2 may have to wait for a usable log file to become available.
    - There is, potentially, a greater likelihood of data loss due to log media failures because of less frequent archiving.
    - Could incur large crash recovery times depending upon the `softmax` database configuration parameter value.
- With a smaller log file size and a larger number of primary logs, the converse applies:
  - The main *advantages* are as follows:
    - Less data loss in log file shipping to remote locations in support of disaster recovery scenarios.
    - With archive logging, less likelihood of waits for a usable log to become available.
    - Less likelihood of data loss due to log media failures because of the occurrence of more frequent archiving.
  - The *disadvantages* are as follows:

- Higher switching costs as one log fills up.
- Higher log file allocation costs with archive logging.
- Higher performance costs with archive logging, since it involves more work on the part of the database manager.

Note: The maximum log file size is 262144 4 K pages on UNIX and WIndows.

The default is 250 4 K pages on Windows, and 1000 4 K pages on UNIX.

Best practices

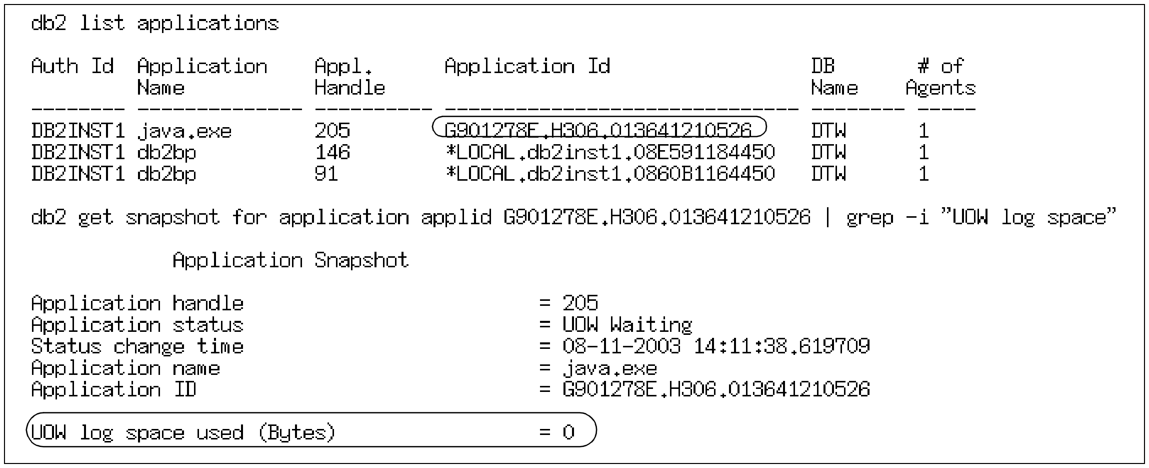
We recommend the following best practices for primary and secondary logs and size of the log files:

1. There is no specific rule of thumb for determining the number of primary and secondary logs for a given workload.

   The total active log space ((**`logprimary`** + **`logsecond`**) x **`logfilsiz`**) should accommodate all the log records generated by the longest running transaction, as well as the log records generated by all transaction executed within the interval of the longest running transaction. Preferably, the total active log space should be supported by primary logs only, since secondary logs are meant to be used as a safety valve to avoid log full conditions.

   Without a means to compute the required total active log space, you should assume your available disk log space is the same as the total active log space, and then monitor usage to tune the **`logprimary`**,**`logsecond`**, and **`logfilsiz`** parameters for your particular environment.

   For an existing system, you can determine the log space taken up by a single unit-of-work (UOW) as shown in Figure 3-22 on page 215. This involves obtaining the **`Application Id`** or **`Appl.Handle`** from the **`db2 list applications`** command, and then using this **`Application Id`** for an application snapshot to determine the amount of log space consumed at the UOW level. By collecting UOW log space (**`uowlogspace`**) consumed for the transaction generating the most log records (performs the most updates) and estimating the number of transactions (**`numtrans`**) executed in the longest running transaction interval, you can roughly estimate the total active log space required as being equal to (**`uowlogspace x numtrans`**).



```
db2 list applications

Auth Id  Application    Appl.      Application Id                 DB       # of
         Name           Handle                                    Name     Agents
-------- -------------- ---------- ------------------------------ -------- -----
DB2INST1 java.exe       205        G901278E.H306.013641210526     DTW      1
DB2INST1 db2bp          146        *LOCAL.db2inst1.08E591184450   DTW      1
DB2INST1 db2bp          91         *LOCAL.db2inst1.0860B1164450   DTW      1

db2 get snapshot for application applid G901278E.H306.013641210526 | grep -i "UOW log space"

            Application Snapshot

Application handle                         = 205
Application status                         = UOW Waiting
Status change time                         = 08-11-2003 14:11:38.619709
Application name                           = java.exe
Application ID                             = G901278E.H306.013641210526

UOW log space used (Bytes)                 = 0
```

Figure 3-22 List application and application snapshot for UOW log space used

2. For OLTP and BI environments:
   - Archive logging is recommended. Circular logging may be appropriate for BI environments that are purely read-only.
   - Choose as large a **logfilsiz** (the current limit is 262144 4 KB pages) as possible within the guidelines of log shipping, archiving and crash recovery time considerations mentioned earlier.
   - Assume 90% of available log space to be reserved for primary logs, and the rest (10%) for secondary logs. Depending upon the file size chosen (**logfilsiz**), the **logprimary** value is computed as ((90% * available log space)/**logfilsiz**) rounded up to an integer value. Ensure that the value of **logprimary** is at least greater than 3 to avoid wait for usable logs due to archiving problems. Choose an appropriate value for **logsecond** based on the remaining available space (10%) and **logfilsiz**.

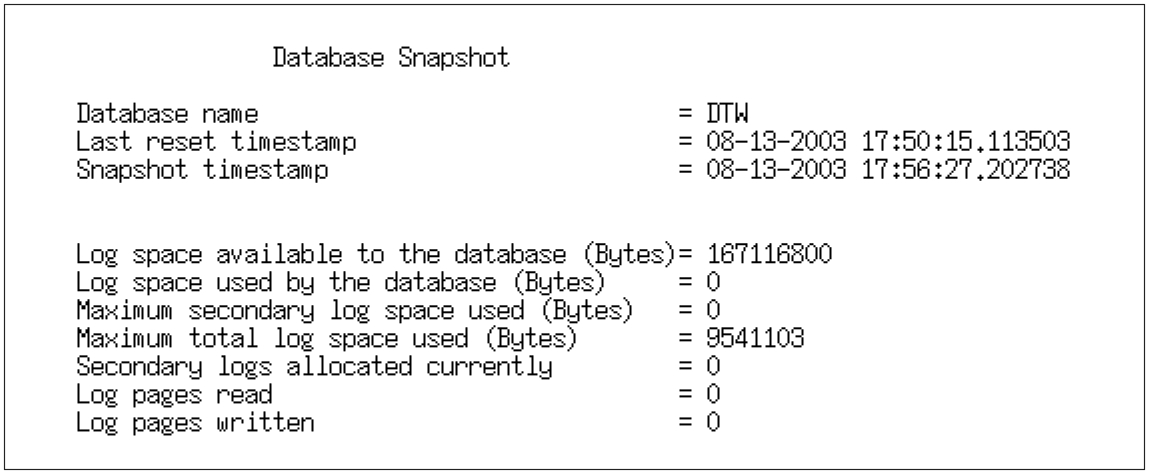
Performance monitoring metrics

There are a number of performance elements captured by the snapshot monitor that can be monitored to tune the **logprimary**, **logsecond**, and **logfilsiz** parameters. The two main ones described here are the database snapshot and the administration notification log, as follows:

1. Database snapshot

   Figure 3-23 on page 216 shows relevant fields that are the output of database snapshot information for the DTW database obtained via the following command:

   ```
   db2 get snapshot for database on DTW
   ```



```
              Database Snapshot

Database name                              = DTW
Last reset timestamp                       = 08-13-2003 17:50:15.113503
Snapshot timestamp                         = 08-13-2003 17:56:27.202738


Log space available to the database (Bytes)= 167116800
Log space used by the database (Bytes)     = 0
Maximum secondary log space used (Bytes)   = 0
Maximum total log space used (Bytes)       = 9541103
Secondary logs allocated currently         = 0
Log pages read                             = 0
Log pages written                          = 0
```

Figure 3-23   Command line database snapshot containing log information

Important: All the fields in the Snapshot Monitor whether they are water marks, counters or gauges, should be monitored over many representative intervals spread over a few weeks, to detect consistent trends before reacting with configuration changes.

Note that counter type monitoring elements should be reset at the beginning of each monitoring interval by issuing the **`RESET MONITOR`** command.

The fields of interest are:

– `Log space available to the database (Bytes)` is a gauge and identifies the amount of active log space in the database that is not being used by uncommitted transactions. If this value goes down to zero, message `SQL0964N` is returned.

– `Log space used by the database (Bytes)` is a gauge and identifies the total amount of active log space currently used in the database.

– `Maximum secondary log space used (Bytes)` is a high water mark.

– `Maximum total log space used (Bytes)` is also a high water mark and includes space consumed by both primary and secondary logs.

– `Secondary logs allocated currently` is a gauge representing the number of secondary logs allocated at that point in time.

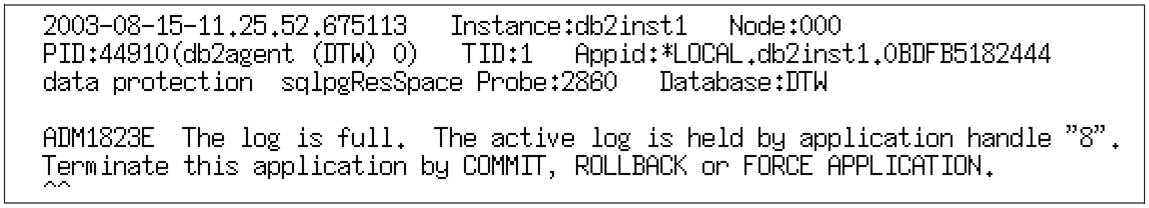
These fields should be evaluated in conjunction with each other to determine whether a configuration change is required.

- Consistent secondary log allocations and a high water mark for the secondary log space used vis-a-vis the total log space used indicates that performance could be improved by:
  - Increasing the number of primary log files (**`logprimary`**).
  - Increasing the size of the log file (**`logfilsiz`**).
  - Issuing more commits in application programs, which tends to reduce the size of the active log space, which in turn makes log files available.

    Note: This measure requires application development cooperation.
- Consistently low levels of available active log space as identified by `Log space available to the database (Bytes)` vis-a-vis `Log space used by the database (Bytes)` indicates a need to increase **`logprimary`**, **`logfilsiz`**, and possibly issue more commits in application programs as described earlier.

2. **Administration notification log**

   Figure 3-24 shows the contents of the administration notification log, which contains messages about log full conditions.



```
2003-08-15-11.25.52.675113   Instance:db2inst1   Node:000
PID:44910(db2agent (DTW) 0)   TID:1   Appid:*LOCAL.db2inst1.0BDFB5182444
data protection  sqlpgResSpace Probe:2860   Database:DTW

ADM1823E  The log is full.  The active log is held by application handle "8".
Terminate this application by COMMIT, ROLLBACK or FORCE APPLICATION.
```

Figure 3-24 Notification log displaying log full condition

Consistent occurrences of log full conditions also indicate that performance could be improved by increasing **`logprimary`** and **`logfilsiz`**, and issuing more commits in application programs as described earlier.

Attention: If additional disk space is not available to increase **`logprimary`** or **`logfilsiz`**, then consider enabling infinite log space by setting **`logsecond`** -1. While this setting may degrade performance of rollback and crash recovery, it avoids the log full condition.

logbufsz db cfg parameter

Having a large buffer size is generally beneficial since it reduces the number of times a buffer is written to the disk due to log buffer full conditions. A large buffer size is also minimizes the number of reads from disk in transaction rollback situations.

However, if the application workload is such that log writes are triggered more often due to reasons other than log buffer full conditions, and there are infrequent transaction rollback situations, then the memory utilized by **`logbufsz`** in the database heap (**`dbheap`** database configuration parameter) could well be better served elsewhere.

Best practices

We recommend the following best practices for **`logbufsz`**:

1. For OLTP environments, choose **`logbufsz`** of 1024 4 K pages, since the default value of 8 4 K pages is insufficient.
2. For BI environments, choose **`logbufsz`** of 512 4 K pages, since the default value of 8 4 K pages is insufficient.

Performance monitoring metrics

The purpose of monitoring is to determine whether the **`logbufsz`** value is too small (thereby causing unnecessary I/Os during rollback), or too big (thereby wasting valuable memory in the database heap).

- ► To determine if the value is too small, use the **`get snapshot for database`** command to capture log reads information as shown in Figure 3-23 on page 216.

  Once again, it is important to gather average, maximum and minimum values of monitored data over extended periods in order detect consistent trends. Avoid making configuration changes before any such trends have been identified.

  The fields of interest are:

  – `Log pages read`, which is a counter that identifies the number of log pages read by the logger process.

  – `Log pages written`, which is also a counter that identifies the number of log pages written to disk by the logger. However, this is *not* equivalent to the number of pages generated by the logger process. When log pages are written to disk, the last page may not be full.

    In such cases, the partial log page remains in the log buffer, and additional log records are written to the page. Therefore, the same log page may be written to disk by the logger process more than once.

  Ideally, the `Log pages read` should be zero, or the ratio of `Log pages read` to `Log pages written` should be very low. If not, consider increasing the **`logbufsz`** and the value of the database configuration parameter **`dbheap`** by the same amount.

- ► There is no simple way to determine if **`logbufsz`** is too big other than through a trial and error method.

If `Log pages read` is almost consistently zero, or the ratio of `Log pages read` to `Log pages written` is very low, consider reducing **logbufsz** in steps; that is, measuring the indicators described at each step, until performance begins to degrade. The optimal value for **logbufsz** is probably the value configured just prior to the manifestation of performance degradation.

mincommit database configuration parameter

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; log I/Os will occur less frequently and more log records are written each time the log buffer is flushed.

Setting this parameter to a value other than 1 (which is the default) can 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.

When commit grouping is in effect, application commit requests are held (wait) until either one second has elapsed or the number of commit requests equals the value of this parameter. Therefore, in lightly loaded systems with few commits, a **mincommit** value greater than 1 can add a response time overhead of up to one second to applications, which might be unacceptable for very short transactions. In most cases, the default value should be adequate.

This parameter is configurable online.

Best practices

We recommend the following best practices for setting the **mincommit** parameter value:

1. For high throughput/performance OLTP systems, consider raising the **mincommit** value to other than 1 in very small increments, using trial and error techniques to arrive at an optimal value.

   For an existing system, you may determine the transaction throughput by capturing database snapshots, as shown in Figure 3-25 on page 220, during many representative measurement intervals spread over a few weeks to determine consistent trends. Perform a **`db2 reset monitor for database <db-alias>`** at the start of the measurement interval resets all the counters, followed by a **`db2 get snapshot for <db-alias>`** at the end of the measurement interval.

```
db2 get snapshot for database on DTW


              Database Snapshot

Database name                              = DTW
Last reset timestamp                       = 08-14-2003 17:11:57.600672
Snapshot timestamp                         = 08-14-2003 17:12:11.286142


Commit statements attempted                = 12
Rollback statements attempted              = 18

Log space used by the database (Bytes)     = 3990
```

Figure 3-25 Database snapshot showing the commits and rollback counts

The total number of transactions in the measurement interval is equal to (`Commit statements attempted` + `Rollback statements attempted`).

The measurement interval is equal to (`Last reset timestamp` - `Snapshot timestamp`).

The transaction throughput is the ratio of (`Commit statements attempted` + `Rollback statements attempted`) and (`Last reset timestamp` - `Snapshot timestamp`).

If **`mincommit`** is increased, consider increasing **`logbufsz`** as well to avoid having a write of the buffer triggered prematurely by a log buffer full condition instead of the **`mincommit`** threshold being tripped during these transaction intensive periods. The **`logbufsz`** may be computed as follows:

`logbufsz = mincommit x (average UOW log space used by a transaction)`

Figure 3-22 on page 215 describes the method for determining the log space consumed by a UOW.

2. For most other OLTP and BI systems, the default value of 1 should be adequate.

Performance monitoring metrics

An optimal **`mincommit`** value for a given workload environment reduces the number of writes to disk without imposing unacceptable overheads of response time waits on critical transactions.

The database snapshot of snapshot monitor shown in Figure 3-23 on page 216 displays a count of the number of `Log pages written`; this field can be monitored with different **`mincommit`** values over many representative measurement intervals spread over a few weeks to detect consistent trends to identify the **`mincommit`** value writing the fewest log pages. Simultaneously, response time measurements of critical transactions for each of these **`mincommit`** values has to be conducted as well.

The optimal **`mincommit`** value can be determined by juxtaposing the acceptable response time measurements of critical transactions with the lowest log pages written for the range of **`mincommit`** values configured. The optimal value is not necessarily the one writing the fewest log records.

Note: Transaction response time measurements have to be obtained using appropriate non-DB2 tools.

blk_log_dsk_full db cfg

This database configuration parameter can be set to prevent disk full errors from causing DB2 to stop processing (`ADM1826E`) when it cannot create a new log file in the active log directory. DB2 will attempt to create the log file at 5-minute intervals until it succeeds; a message (`ADM1828C`) is written to the administration notification log at each attempt.

Setting this parameter to **`YES`** causes *all* applications to appear to hang until the log disk full condition is resolved, while a **`NO`** setting causes the transaction that receives the log disk full error to fail and be rolled back. In some situations, the database will come down if a transaction causes a log disk full error.

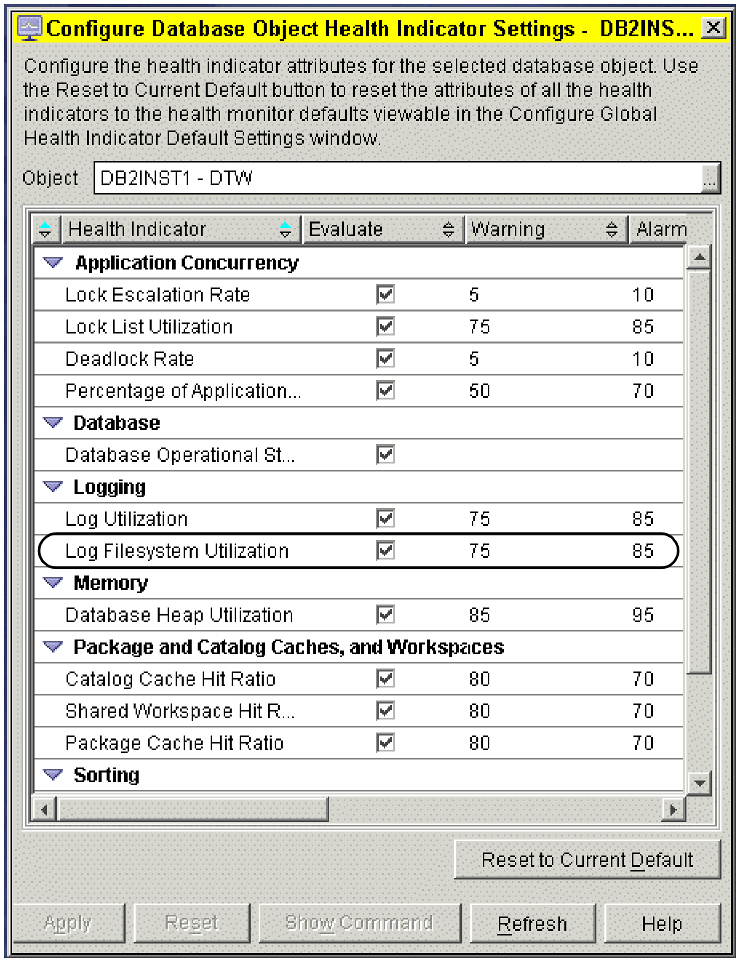
Best practices

For both OLTP and BI environments, the recommendation is to set **`blk_log_dsk_full`** to **`YES`** so as to potentially avoid bringing down DB2 in the case of a log disk full condition.

Performance monitoring metrics

Messages `ADM1826E DB2 can not continue because the disk used for logging is full` and `ADM1828C DB2 will attempt to create the log file again in 5 minutes` in the administration notification log indicate a log disk full condition.

Impending log disk full conditions may also be monitored via the log filesystem utilization health indicator of the Health Center, as shown in Figure 3-26 on page 222.



Configure Database Object Health Indicator Settings - DB2INS...

Configure the health indicator attributes for the selected database object. Use the Reset to Current Default button to reset the attributes of all the health indicators to the health monitor defaults viewable in the Configure Global Health Indicator Default Settings window.

Object: DB2INST1 - DTW

Health Indicator	Evaluate	Warning	Alarm
Application Concurrency			
Lock Escalation Rate	✓	5	10
Lock List Utilization	✓	75	85
Deadlock Rate	✓	5	10
Percentage of Application...	✓	50	70
Database			
Database Operational St...	✓		
Logging			
Log Utilization	✓	75	85
Log Filesystem Utilization	✓	75	85
Memory			
Database Heap Utilization	✓	85	95
Package and Catalog Caches, and Workspaces			
Catalog Cache Hit Ratio	✓	80	70
Shared Workspace Hit R...	✓	80	70
Package Cache Hit Ratio	✓	80	70
Sorting			

Figure 3-26 Log filesystem utilization health indicator in Health Center

Log filesystem utilization is measured as the percentage of space consumed in the active log’s filesystem. Alerts can be set and generated if the amount of free space is falls below a given threshold.

Log filesystem utilization is calculated as follows:

```
Log file utilization = (fs.log_fs_used / fs.log_fs_total)*100
```

where fs is the file system on which the log resides.

3.4.3 Monitor switch settings

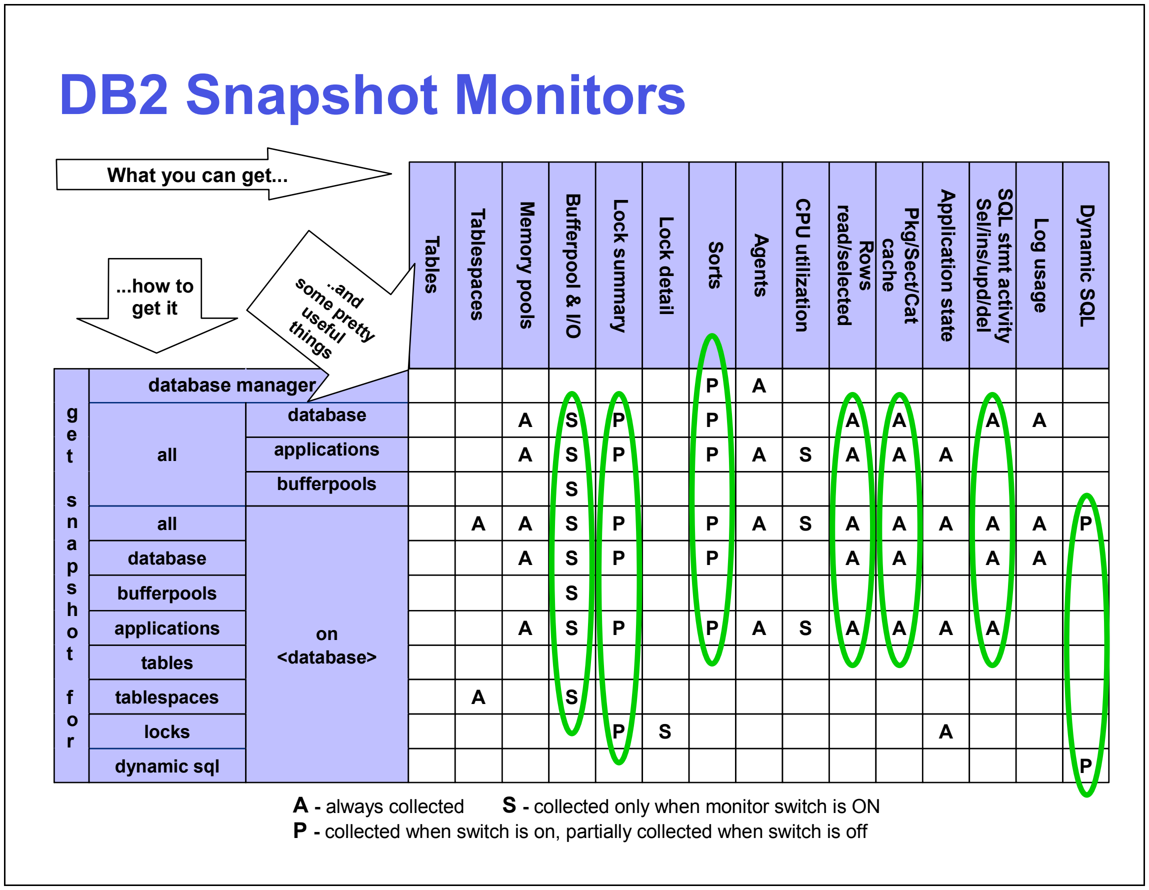
DB2 provides a number of monitoring capabilities for problem diagnosis, as well as tuning the performance of the system, as described in 2.6, “Performance monitoring facilities” on page 66.

The information collected by the Snapshot Monitor of the Database System Monitor in particular is controlled by a set of monitor switches defined in the database manager configuration file; these switches and their default settings are described in Table 3-11.

Attention: There is a considerable amount of basic monitoring data that is not under monitor switch control, and will always be collected regardless of switch settings. Figure 3-27 on page 224 shows the monitoring data collected in relation to monitor switch settings.

Table 3-11 Snapshot Monitor switches

Monitor switch	Default	Description
DFT_MON_BUFPOOL	OFF	Buffer pool activity information, such as number of reads and writes, and time taken
DFT_MON_LOCK	OFF	Lock wait times and deadlock-related information
DFT_MON_SORT	OFF	Sorting information, such as number of heaps used and sort performance
DFT_MON_STMT	OFF	SQL statement information, such as start/stop time, and statement identification
DFT_MON_TABLE	OFF	Table activity information, such as rows read and written
DFT_MON_TIMESTAMP	ON	Times and timestamp information
DFT_MON_UOW	OFF	Unit of work information, such as start/end times and completion status



DB2 Snapshot Monitors

What you can get...
...how to get it
..and some pretty useful things

get snapshot for			Tables	Tablespaces	Memory pools	Bufferpool & I/O	Lock summary	Lock detail	Sorts	Agents	CPU utilization	Rows read/selected	Pkg/Sect/Cat cache	Application state	SQL stmt activity Sel/ins/upd/del	Log usage	Dynamic SQL
database manager									P	A							
all	database				A	S	P		P			A	A		A	A	
all	applications				A	S	P		P	A	S	A	A	A			
all	bufferpools					S											
all		on <database>		A	A	S	P		P	A	S	A	A	A	A	A	P
database		on <database>			A	S	P		P			A	A		A	A	
bufferpools		on <database>				S											
applications		on <database>			A	S	P		P	A	S	A	A	A	A		
tables		on <database>															
tablespaces		on <database>		A		S											
locks		on <database>					P	S						A			
dynamic sql		on <database>															P

A - always collected **S** - collected only when monitor switch is ON
P - collected when switch is on, partially collected when switch is off

Figure 3-27 DB2 Snapshot Monitor syntax and data collection

Note: Event Monitors are not affected by monitor switches in the same way as snapshot monitoring applications. When an Event Monitor is defined, it automatically turns `ON` the instance level monitor switches required by the specific event types.

Refer to "Snapshot Monitor" on page 69 for more information about monitor switches including setting them, taking snapshots, and other general considerations.

Performance considerations

There are overheads associated with collecting Database System Monitor, including collecting the data when the monitor switches are set, as well as the processing cost of frequently retrieving this information via the **`get snapshot`** and **`flush event monitor`** commands.

Each monitor switch setting imposes a certain overhead that is dependent upon the nature of the workload. The overheads are predominantly related to CPU consumption rather than waits due to concurrency issues.

Typical overheads are as follows:

- All switches set - approximately 5 to 10%.
- **`DFT_MON_LOCK`** setting imposes an overhead of between 1 to 3%, depending upon the frequency of snapshot requests.
- **`DFT_MON_STMT`** setting with dynamic SQL workloads imposes an overhead of between 5 to 10%, proportional to statement throughput. Timestamp switch setting needs to be considered in the overhead experienced.
- **`DFT_MON_TIMESTAMP`** setting (default is **`ON`**) overhead can becoming significant when CPU utilization approaches 100%. When this setting is **`ON`**, the database manager issues timestamp operating system calls when determining time or timestamp-related monitor elements. When this setting is **`OFF`**, the overhead of other switch settings is greatly reduced.

Important: Since the overhead associated with each monitor switch setting is completely dependent upon the workload and the frequency of snapshot requests, you should determine the overheads in your specific environment through careful measurements. The overheads listed earlier are merely provided as guidelines to be used prior to detailed measurement in your environment.

Best practices

We recommend the following best practices for monitor switch settings for *routine monitoring* in OLTP and BI environments. Routine monitoring overhead should typically not exceed 5%:

1. For OLTP environments:
   - Set all switches to **`ON`** except **`DFT_MON_STMT`** and **`DFT_MON_LOCK`**.
   - The frequency of snapshot requests should be low for lightly loaded systems (say, every 60 seconds), and higher for highly dynamic systems (say, every 15 seconds).
   - Let **`DIAGLEVEL`**[6] default to 3.
   - Let the **`NOTIFYLEVEL`**[7] default to 3.

[6] Specifies the type of diagnostic errors recorded in the db2diag.log

– Let **`HEALTH_MON`**[8] default to **`ON.`**

> **Note:** **`DIAGLEVEL`**, **`NOTIFYLEVEL`**, and **`HEALTH_MON`** are described in 2.6, "Performance monitoring facilities" on page 66.

2. For BI environments:
   - Set all switches to **`ON`** except **`DFT_MON_STMT`**, **`DFT_MON_UOW`**, and **`DFT_MON_LOCK`**.
   - Frequency of snapshot requests should be low for lightly loaded systems (every 300 seconds), and higher for highly dynamic systems (every 60 seconds).
   - Let **`DIAGLEVEL`** default to 3.
   - Let the **`NOTIFYLEVEL`** default to 3.
   - Let **`HEALTH_MON`** default to **`ON.`**

> **Attention:** These setting should be evaluated in the context of overheads incurred in your specific environment.

3.4.4 Connection considerations

When an application wants to perform requests against a database, it first has to connect to the database before it can issue SQL requests against the data. A connection involves associating a db2agent (and subagent db2agntp, if appropriate) on the database server with the application that services requests on behalf of the client. An agent facilitates the operations between the application and the database. The flow of a typical application's request is described in detail in 2.3, "Single user transaction/query flow" on page 45.

Each agent operates with its own private memory and shares database manager and database global resources such as the buffer pool with other agents. Being an operating system process (thread), each agent may also consume disk space (such as paging) and CPU cycles, and increases the potential for contention for shared resources. In addition, the cost of creating and destroying processes may affect the throughput of the overall system. Therefore, the workload and environment of the database server needs to be managed to ensure performance objectives are met.

[7] Specifies the type of administration notification messages that are written to the administration notification log.
[8] Specifies whether the DB2 instance, its associated databases, and database objects should be monitored using the Health Center's health indicators.

In order to manage the workload on a database server, and improve the efficiency of DB2 agent processes, DB2 provides the following agent related parameters to control resource consumption and contention on the database server.

1. Database manager configuration parameters
   - max_connections
   - maxagents
   - max_coordagents
   - maxcagents
   - num_poolagents
   - num_initagents
2. Database configuration parameter
   - maxappls

Each of these parameters are reviewed in turn, with a brief description of their function, performance considerations associated with their use, best practices for using them, and monitoring metrics for evaluating their effectiveness.

max_connections

This parameter controls the maximum number of applications that can be connected to the DB2 instance (it was called **max_logicagents** in DB2 UDB Version 7).

Typically, each application is assigned a coordinator agent. When the connection concentrator is enabled, a logical agent entity is also created prior to the coordinator agent being assigned as described in 2.3, "Single user transaction/query flow" on page 45. The intent of the connection concentrator is to reduce the server resources per client application to a point a DB2 Connect gateway can handle more than tens of thousands of client connections. The connection concentrator may improve performance since each EDU consumes additional memory when the number of connections increase, and context switching between agents results in additional overhead.

The default is -1 which corresponds to the value of the **max_coordagents** parameter.

Performance considerations

If the **max_connections** parameter is set too low, then applications will fail with an **SQL1226N** error code when they attempt to connect to the database resulting in the perception of poor performance. On the other hand, setting it too high may flood the database server with requests, causing resource utilization and contention problems and also resulting in performance problems.

Best practices

We recommend the following best practices for `max_connections`:

1. For 2-tier OLTP environments with many simultaneous user connections, enable the connection concentrator by setting `max_connections` greater than `max_coordagents.`

   For 3-tier environments, the Web Application Server (such as WebSphere Application Server) performs connection pooling, thereby eliminating the need for enabling connection concentrator.

2. For other OLTP and BI environments, let this value default.

Monitoring performance metrics

An underconfigured value for this parameter will result in user complaints due to applications receiving the SQL1226N return code due to connection failures. In such cases, a trial and error method should be adopted to find the optimum value for `max_connections`. Increase this value in incremental steps until the application connection failures fall to an acceptable number without causing other performance problems arising out of increased contention for shared resources.

An overconfigured value will only cause problems if other controlling parameters such as `maxagents`, `max_coordagents`, and `maxcagents` do not exercise sufficient controls on the number of processes allowed to consume resources. If they do not, then the database server will likely get flooded with agent processes and cause performance problems due to increased contention for scarce resources.

maxagents

This parameter indicates the maximum number of database manager agents, whether coordinator agents or subagents, available at any given time to accept application requests.

Note: To specifically limit the number of coordinating agents, use the `max_coordagents` parameter.

The `maxagents` 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.

The default value is 200.

Performance considerations

An underconfigured value for this parameter will save memory but reduce the throughput of a system with adequate system resources, while an overconfigured value in a resource-constrained system can result in significant contention for system resources, resulting in poor performance.

Best practices

We recommend the following best practices for **`maxagents`**:

1. For new systems, use the Configuration Advisor as described in 2.5.1, “Configuration Advisor and AUTOCONFIGURE” on page 54 to estimate the value for **`maxagents`**.
2. For OLTP production environments, only a single database is typically associated with an instance, and this parameter value should be equal to **`maxappls`**.

   If there is more than one concurrently active database associated with this instance (the **`numdb`** database manager configuration parameter specifies the maximum allowable number of databases, while the **`list db directory`** command identifies the actual number of databases defined), then the safest course for **`maxagents`** is to choose a value that is the product of **`numdb`** and the largest value of **`maxappls`** of all the databases.

   Note: The assumption is that the database manager configuration parameter **`INTRA_PARALLEL`** is disabled for OLTP environments.

3. For BI environments, which may have the database manager configuration parameter **`INTRA_PARALLEL`** enabled, this value should include the subagents required to process application requests.

   If more than one database is associated with an instance, then the value computed above should be multiplied by the **`numdb`** value for this instance.

Performance monitoring metrics

Example 3-13 shows a select list of fields from the **`get snapshot for dbm`** command.

Example 3-13 Database manager snapshot

```
db2 => get snapshot for dbm

            Database Manager Snapshot
......
Remote connections to db manager               = 0
Remote connections executing in db manager     = 0
Local connections                              = 4
Local connections executing in db manager      = 0
Active local databases                         = 1

High water mark for agents registered          = 5
High water mark for agents waiting for a token = 0
Agents registered                              = 5
Agents waiting for a token                     = 0
Idle agents                                    = 0
......
Agents assigned from pool                      = 2
Agents created from empty pool                 = 7
Agents stolen from another application         = 0
High water mark for coordinating agents        = 5
Max agents overflow                            = 0
Hash joins after heap threshold exceeded       = 0

Total number of gateway connections            = 0
Current number of gateway connections          = 0
Gateway connections waiting for host reply     = 0
Gateway connections waiting for client request = 0
Gateway connection pool agents stolen          = 0
.....
```

Important: All the fields in the snapshot monitor, whether they are water marks, counters or gauges, should be monitored over many representative intervals spread over a few weeks, to detect consistent trends before reacting with configuration changes.

Note that counter-type monitoring elements should be reset at the beginning of each monitoring interval by issuing the **`RESET MONITOR`** command.

The following fields are relevant to tuning the **`maxagents`** parameter.

- The `High water mark for agents registered` field is a water mark that identifies the maximum number of agents (coordinator and subagents) at the same time since it was started.
- The `Max agents overflow` field is a gauge that identifies the number of times a request to create a new agent was received when the **`maxagents`** value had already been reached.

If the `High water mark for agents registered` field value equals **`maxagents`**, then **`maxagents`** is possibly under configured, and the `Max agents overflow` field should be monitored for frequent non-zero values before raising the **`maxagents`** value to increase system throughput in an environment that is not resource-constrained.

If the `High water mark for agents registered` field value is much lower than **`maxagents`**, then **`maxagents`** is possibly overconfigured and could be reduced in a resource-constrained environment.

max_coordagents

This parameter specifies the maximum number of coordinating agents that can exist at one time on a DB2 instance. One coordinating agent is required for each local or remote application that connects to the database or attaches[9] to the instance. This parameter can be used to control the load on the system.

The default value is -1, which corresponds to (**`maxagents - num_initagents`**) when the database manager configuration parameter **`INTRA_PARALLEL`** is enabled, and **`maxagents`** when **`INTRA_PARALLEL`** is disabled.

Performance considerations

Here again, an underconfigured value for this parameter will save memory but reduce the throughput of a system with adequate system resources, while an overconfigured value in a resource-constrained system can result in significant contention for system resource, resulting in poor performance.

Best practices

We recommend the following best practices for **`max_coordagents`**:

1. For new systems, use the Configuration Advisor as described in 2.5.1, “Configuration Advisor and AUTOCONFIGURE” on page 54 to estimate the value for **`max_coordagents`**.
2. Assuming that the value of **`maxagents`** has been set correctly as discussed in “maxagents” on page 228, let this parameter default for both OLTP and BI environments.

Performance monitoring metrics

Referring back to Example 3-13 on page 229, the field of interest in tuning **`max_coordagents`** is `High water mark for coordinating agents`, which is a water mark for concurrently executing applications.

If the `High water mark for coordinating agents` field value equals **`max_coordagents`**, then **`max_coordagents`** is possibly underconfigured, and its value should be raised to increase system throughput in an environment that is not resource-constrained. In a resource-constrained environment, consider setting **`maxcagents`** to a lower value to reduce the load on the system instead of lowering `the value of` **`max_coordagents.`**

[9] Requests that require an instance attachment include CREATE DATABASE, DROP DATABASE, and Database System Monitor commands,

If the **`High water mark for coordinating agents`** field value is much lower than **`max_coordagents`**, then **`max_coordagents`** is possibly overconfigured and could be reduced in a resource-constrained environment.

maxcagents

This parameter limits the maximum number of database manager agents that can be concurrently executing a database manager transaction. Whenever an DB2 agent is assigned work, it attempts to obtain a token or permission to process the transaction. This parameter controls the number of tokens available to the database manager.

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.

The default value is -1, which corresponds to the value of **`max_coordagents`**.

Performance considerations

This parameter is used to control the load on the system during periods of high simultaneous application activity. This is particularly useful in CPU, disk, and memory-constrained environments with a large number of connections, where this parameter can limit high simultaneous activity that could cause excessive operating system paging and performance degradation. However, setting this parameter can result in long wait times and concurrency problems.

Best practices

We recommend the following best practices for **`maxcagents`**:

1. For both OLTP and BI environments, let the value default to **`max_coordagents`**.

Performance monitoring metrics

Important: All the fields in the snapshot monitor whether they are water marks, counters or gauges, should be monitored over many representative intervals spread over a few weeks, to detect consistent trends before reacting with configuration changes.

Note that counter-type monitoring elements should be reset at the beginning of each monitoring interval by issuing the **`RESET MONITOR`** command.

Example 3-13 on page 229 shows the following fields of interest for tuning **`maxcagents:`**

- `Agents waiting for a token`, which is a gauge for the number of agents waiting to get a token to execute a database transaction.
- `Remote connections executing in db manager`, which is a gauge of the number of remote applications that are currently connected to a database and performing a unit of work.
- `Local connections executing in db manager`, which is a gauge of the number of local applications that are currently connected to a database and performing a unit of work

Consider the following approaches for tuning **`maxcagents`**:

- If the `Agents waiting for a token` field reports frequent non-zero values, consider raising the value of **`maxcagents`** to increase system throughput in non-resource constrained environments.
- Use benchmarking techniques to increase the value only if high performing, high concurrency applications is causing problems to other users/applications in the system and also when the system is paging high.
- Monitor (`Remote connections executing in db manager` + `Local connections executing in db manager`) which is the number of tokens being executed in the database manager to determine their value relative to **`maxcagents`**, which can then be used to set its value.

num_poolagents

This parameter determines the maximum size of the idle agent pool. As mentioned earlier, there is an overhead associated with destroying and recreating an agent process whenever applications disconnect and connect to the database. To minimize this overhead, DB2 keeps the idle agents in an agent pool to be reused by other applications.

The processing involved when an agent finishes processing a request depends upon whether connection concentrator is disabled or not as follows:

- With connection concentrator disabled, if more agents are created than indicated by **`num_poolagents`**, then agents will be terminated when they finish executing their current request instead of returning them to the pool.
- With connection concentrator enabled, **`num_poolagents`** 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.

The maximum agent pool size is equal to **`maxagents.`**

The default value is -1, which with local and remote clients corresponds the larger of the following:

```
((maxagents/50) x max_querydegree) or (maxagents - max_coordagents)
```

If **`num_poolagents`** is zero, agents will be created as needed, and may be terminated when they finish executing their current request.

Performance considerations

An underconfigured value for this parameter will result in agents being terminated, which could result in new application requests having to incur the cost of agent creation and thereby incur performance degradation. An overconfigured value results in wasted memory resources.

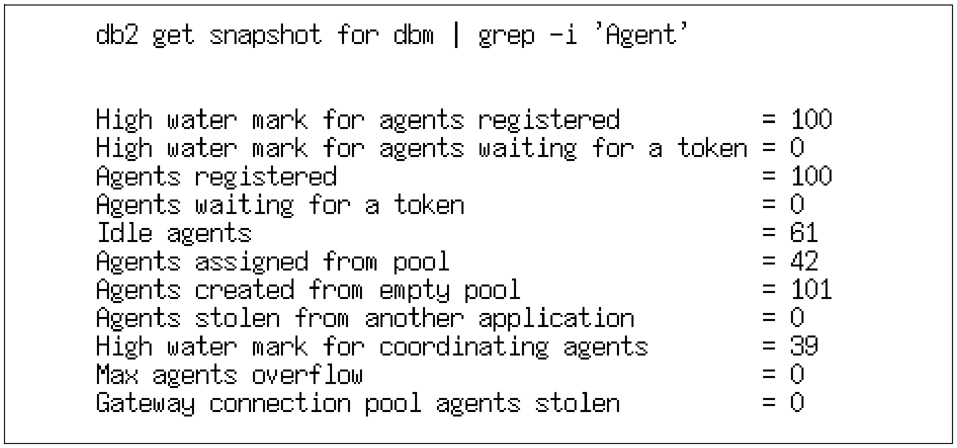
Best practices

We recommend the following best practices for tuning **`num_poolagents`**:

1. For OLTP environments, the objective should be to stay connected for optimal performance. If connections are constantly created and destroyed, a larger agent pool should be defined to avoid the costs associated with the frequent creation and termination of agents.
2. BI environments tend to have few concurrent queries, and each query tends to be long running and therefore be less impacted by the cost of agent creation. Set **`num_poolagents`** to a smaller value than in the case of OLTP environments to avoid having an agent pool that is full of idle agents.

Performance monitoring metrics

Figure 3-28 lists selected fields from the **`get snapshot for dbm`** command.



```
db2 get snapshot for dbm | grep -i 'Agent'


High water mark for agents registered          = 100
High water mark for agents waiting for a token = 0
Agents registered                              = 100
Agents waiting for a token                     = 0
Idle agents                                    = 61
Agents assigned from pool                      = 42
Agents created from empty pool                 = 101
Agents stolen from another application         = 0
High water mark for coordinating agents        = 39
Max agents overflow                            = 0
Gateway connection pool agents stolen          = 0
```

Figure 3-28 Database manager snapshot on Agent information

Important: All the fields in the snapshot monitor, whether they are water marks, counters or gauges, should be monitored over many representative intervals spread over a few weeks, to detect consistent trends before reacting with configuration changes.

Note that counter-type monitoring elements should be reset at the beginning of each monitoring interval by issuing the **`RESET MONITOR`** command.

The following fields are of interest for tuning **`max_poolagents:`**

- `Idle agents` is a gauge of the number of agents that are currently unassigned to an application and are therefore "idle".
- `Agents assigned from pool` is a counter that records the number of agents assigned from the agent pool.
- `Agents created from empty pool` is a counter that records the number of agents created because the agent pool was empty. It includes the number of agents started at DB2 startup (**`num_initagents`**).

Consider modifying the value of **`num_poolagents`** in the following cases:

- If the `Idle agents` field is consistently zero or very low, increase the value of **`num_poolagents`** to minimize agent creation and destruction costs.

  If the `Idle agents` field is consistently high, decrease the value of **`num_poolagents`** to avoid wastage of system resources.
- The ratio (`Agents created from empty pool/Agents assigned from pool`) is a measure of how often an agent must be created because the pool is empty.

  If this ratio is consistently high, then increase the value of **`num_poolagents`** to minimize agent creation and destruction costs.

  If the ratio is consistently low, then decrease the value of **`num_poolagents`** to avoid wastage of system resources.

num_initagents

This parameter determines the initial number of idle agents that are created in the agent pool (as defined by **`num_poolagents`**) at **`db2start`** time. When the database manager is started, a pool of worker agents is created based on this value, which speeds up the performance for initial queries as they will not incur the cost of creating the agent. The worker agents all begin as idle agents.

The default value is zero.

Performance considerations

The wait time for agent creation is reduced when **`num_initagents`** is greater than zero. As mentioned earlier, agents consume memory and should not be overconfigured.

Best practices

We recommend the following best practices for tuning **`num_initagents`**:

1. For OLTP environments, the objective should be to avoid the costs associated with the initial creation of agents. **`num_initagents`** should therefore be equal to the expected average concurrent workload (DB2 agents and subagents). Ensure that it is less than **`num_poolagents`** so that these agents are not terminated when they have finished their work.
2. BI environments tend to have few concurrent queries and should therefore have a small value for **`num_initagents`**.

Performance monitoring metrics

The same performance monitoring metrics as described in "num_poolagents" on page 233 apply here as well.

maxappls

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.

When an application attempts to connect to a database, but **`maxappls`** has already been reached, an error (**`SQL1040N`**) is returned to the application indicating that the maximum number of applications have been connected to the database.

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.

The default value is **`AUTOMATIC`**.

Performance considerations

Setting **`maxappls`** to **`AUTOMATIC`** can lead to an over commitment resources in constrained environments, resulting in significant performance degradation. Setting **`maxappls`** to a value enables you to control the workload and resource consumption in resource-constrained environments.

The setting of **maxappls** has an impact on the size of the package cache (**pckcachesz**) and the catalog cache (**catalogcache_sz)** if their settings are allowed to default as follows:

- Package cache defaults to 4 times **maxappls**
- Catalog cache defaults to 8 times **maxappls**

maxappls is indirectly dependent on **maxagents** and **max_coordagents.** An application can only connect if there is a DB2 agent available. No new applications can be started if **maxagents** or **max_coordagents** has been reached.

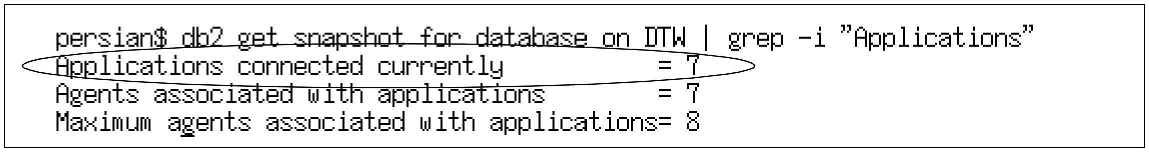
Best practices

We recommend the following best practices for tuning **maxappls**:

1. For an existing system, consider using the Configuration Advisor.
2. Let it default to **AUTOMATIC** unless you have a very resource-constrained environment.
3. When setting this parameter to a value other than **AUTOMATIC**, choose a value that is equal to or greater than the sum of concurrent 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.

Performance monitoring metrics

Figure 3-29 lists selected fields of the **get snapshot for database** command.



```
persian$ db2 get snapshot for database on DTW | grep -i "Applications"
Applications connected currently            = 7
Agents associated with applications         = 7
Maximum agents associated with applications= 8
```

Figure 3-29 Database snapshot - applications connected currently

Important: All the fields in the snapshot monitor, whether they are water marks, counters, or gauges, should be monitored over many representative intervals spread over a few weeks, to detect consistent trends before reacting with configuration changes.

Note that counter type monitoring elements should be reset at the beginning of each monitoring interval by issuing the **RESET MONITOR** command.

The field of interest for tuning **maxappls** is `Applications connected currently`, which is a gauge that indicates the number of applications that are currently connected to the database.

Use this value to set **maxappls** to a specific value to control the consumption of system resources in resource-constrained environments.

3.4.5 Buffer pool considerations

Buffer pools tend to be one of the major components that can have the most dramatic impact on performance, since they have the potential to reduce I/Os. A 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.

A buffer pool is memory used to cache both user and system catalog table and index pages as they are being read from disk, or being modified. A buffer pool is also used as overflow for sorts.

Note: Large objects (LOBs) and long fields (LONG VARCHAR) data are not manipulated in the buffer pool.

In the following subsections, we discuss the following topics:

- Buffer pool flow
- Main characteristics of buffer pools
- Performance considerations
- Best practices
- Performance monitoring metrics

Buffer pool flow

The general processing flow with buffer pools is shown in Figure 3-30 on page 239.

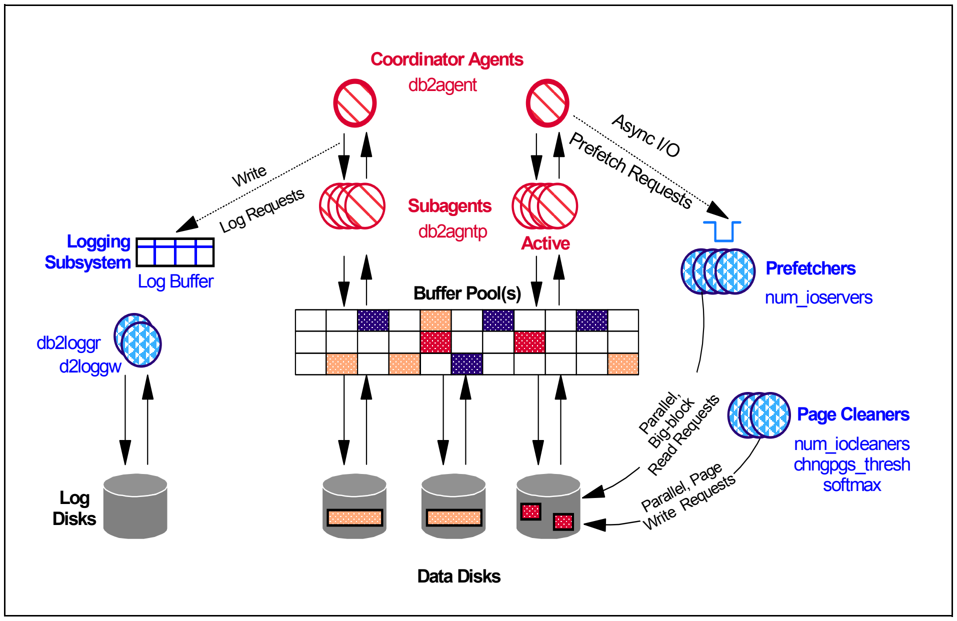


Figure 3-30 Buffer pool flow

The following events may occur with buffer pools during normal processing:

1. An application (db2agent or db2agntp) requests data from the buffer pool, which returns the data to the application if it is found in a page in the buffer pool, and since there is no I/O involved, access is very fast. If the data is not found in a page in the buffer pool, the buffer pool manager retrieves the appropriate page from disk and places it into the buffer pool for the application to retrieve the data. While this I/O is in progress, the application waits, which results in slower access from the application's point of view.

   When the application performs an update to data, a similar process is involved, with access being faster if the required pages are in the buffer pool.

2. Once the pages are in the buffer pool, they remain there until they are replaced.

3. The database manager also attempts to bring pages into the buffer pool in anticipation of their being used by an application using certain processes called *prefetchers* (db2pfchr).

4. DB2 identifies pages in the buffer pool as being in one of the following states:

   – **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.

– **Clean** pages are changed pages that have been written to disk. However, this page 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.

DB2 manages pages in the buffer pool in such a way that it attempts to:

– Ensure that pages required by an application are brought in to the buffer pool before they are requested by an application using the prefetchers (db2pfchr).

> **Note:** The number of prefetchers is controlled by the **`num_ioservers`** database configuration parameter.

Prefetching involves retrieving one or more pages 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 as described in “Prefetch size for the table space” on page 152.

There are two categories of prefetching: sequential prefetch, and list (sequential) prefetch, which we explain as follows:

- *Sequential prefetch*, which reads *consecutive* pages into the buffer pool, using a single I/O operation before the pages are required by the application.

  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.

  For example the following statement, which probably requires a table scan, would be a good candidate for sequential prefetch:

  ```
  SELECT NAME FROM EMPLOYEE
  ```

  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. The **`seqdetect`** database configuration parameter (default is enabled) to control whether the database manager performs sequential detection or not.

For example, with sequential detection enabled, 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.

- *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.

– Ensure that there are always clean pages available in the buffer pool for accommodating new incoming pages, by cleaning pages on a regular basis using the page cleaners (***db2pclnr***). Page cleaners perform I/O as background processes.

Note: The number of page cleaners is controlled by the **`num_iocleaners`** database configuration parameter.

Page cleaners are triggered by the following:

- When the percentage of space occupied by changed pages in any one buffer pool exceeds the value specified by the **`chngpgs_thresh`** database configuration parameter; it specifies the percentage of changed pages (default is 60) in the buffer pool at which the asynchronous page cleaners will be started if they are not currently active. This is also called a dirty page threshold.

  When the page cleaners are triggered, *all* page cleaners are triggered at the same time. Each of these page cleaners will collect a fixed number of dirty pages and start writing them out, one page at a time, until they are all flushed to disk. After a page cleaner has completed flushing its assigned dirty pages, it repeats the process if there are additional dirty pages to flush; otherwise, it becomes dormant until the next page cleaner trigger.

> **Attention:** **`chngpgs_thresh`** is a database configuration parameter, not a buffer pool parameter.

- When the **`softmax`** database configuration parameter limit is exceeded; this specifies a percentage of the size of one primary log file. The default is 100, which means that the database manager will try to keep the number of logs that need to be recovered to one (1) and is used to influence the number of logs required for crash recovery by triggering the page cleaners to ensure that pages older than the specified recovery window are already written to disk. This is also called an *LSN Gap threshold*.
- When a page needs to be stolen synchronously by an application; the application waits while the dirty page is written out to disk by the agent and the required new page is brought in its spot. This is also called *dirty page steals*.

Main characteristics of buffer pools

The main characteristics of buffer pools briefly covered here include:

Memory allocation

- ▸ Dynamic buffer pools
- ▸ Multiple buffer pools of varying page sizes
- ▸ Hidden buffer pools
- ▸ Block-based buffer pools
- ▸ Windows 2000 Address Windowing Extensions (AWE)
- ▸ Extended storage cache on 32-bit platforms
- ▸ Buffer pool overhead

Memory allocation

Memory is allocated for buffer pools when a database is activated via the **`ACTIVATE DATABASE`** command, or when an application first connects to the database. Buffer pools are deallocated when all applications disconnect from the database if the database was not explicitly activated; otherwise, an explicit **`DEACTIVATE DATABASE`** or **`db2stop`** command must be used to shut down the database.

Dynamic buffer pools

Buffer pools can be created, dropped, or resized while the database manager is running. The **`IMMEDIATE`** keyword in the **`ALTER BUFFERPOOL`** and **`CREATE BUFFERPOOL`** statement allows increases in the size of the buffer pool to be effective immediately when memory is available; otherwise, it becomes effective after all the applications have disconnected from the database and the database is reactivated. If you decrease the size of the buffer pool, memory is deallocated at commit time.

Note: Dynamic buffer pool sizes are ideal for varying buffer pool configurations between batch, OLTP and decision support workloads during different processing windows. This enables optimal buffer pool configurations for each processing window.

Multiple buffer pools of varying page sizes

Multiple buffer pool sizes are supported of the same and varying page sizes such as 4 K, 8 K, 16 K and 32 K.

- A default buffer pool named **`IBMDEFAULTBP`** with a 4 K page size is created at database creation. On UNIX, the default size is 1000 pages; on Windows, the default size is 250 pages.

  Note: The maximum size of a buffer pool depends upon the specific platform, and whether it is a 32-bit or 64-bit implementation. Very large buffer pools can be defined on 64-bit platforms.

  64-bit implementations are not constrained by virtual memory addressing limitations and can directly exploit machines with very large real memories.

Hidden buffer pools

DB2 creates four buffer pools of 16 pages, each with a page size of 4 K, 8 K, 16 K and 32K when the first client connects to the database.

These buffer pools are used by DB2 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 is likely to be significantly impacted when this occurs because of the limited size of these buffer pools, but it provides a fail-safe mechanism.

- When the buffer pools cannot be brought up during a database connect due to conditions such as an out-of-memory condition. An `SQL1478W (SQLSTATE 01626)` message is returned to the application, and a message is also written to the administration notification log.

Here again DB2 performance is likely to be significantly impacted by this fail-safe mechanism.

Attention: These buffer pools are hidden from the user, and are not present in the system catalogs or in the buffer pool system files. They cannot be used or altered directly.

However, the **DB2_OVERRIDE_BPF** environment registry variable can be set to change the default value of 16 pages for these buffer pools. Should even a minimal buffer pool of 16 pages not be brought up by the database manager (see the administration notification log for messages), 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 sized buffer pools. This value, if set, overrides the current buffer pool size.

Block-based buffer pools

DB2 supports the concept of block-based buffer pool, which provides block-based I/O for buffer pools using **NUMBLOCKPAGES** and **BLOCKSIZE** parameters in the **CREATE BUFFERPOOL** statement. Block-based buffer pools greatly improve the performance of 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.

Note: By default, 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, and a block-based buffer pool provides this capability. A block-based buffer pool consists of both a page area and a block area, as described here:

- 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 called the block size as defined by **BLOCKSIZE**.

The size of this block-based area is defined by the `NUMBLOCKPAGES` parameter. This area of the buffer pool will be used exclusively for sequential prefetching.

Note: A value of zero for `NUMBLOCKPAGES` disables block I/O.

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.

Windows 2000 Address Windowing Extensions (AWE)

Windows 2000 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.

Attention: The strong recommendation is to implement 64-bit DB2 to fully exploit large memories that are becoming increasingly common in Windows environments, rather than use Windows 2000 AWE with its limitations.

The following limitations apply if AWE support is enabled:

- Extended storage can not be used for any of the buffer pools in the database.
- Buffer pools referenced with this registry variable must already exist in `SYSCAT.SYSBUFFERPOOLS`.
- 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.

Extended storage cache on 32-bit platforms

DB2 supports the concept of extended storage cache on 32-bit platforms that have large quantities of real memory (>> 4 GB). Since 32-bit can only directly access 4 GB of memory, the additional real memory can be configured beyond the virtual addressable memory of 4 GB as an extended storage cache.

Attention: Here again, the very strong recommendation is to implement 64-bit DB2 to fully exploit the large memories available, without suffering the limitations of extended storage cache.

A key limitation of extended storage cache is that any real addressable memory defined as an extended storage cache can *no longer* be used for other purposes, such as a JFS-cache or as process private address space. More system paging might occur by allocating real addressable memory to an extended storage cache, which is extremely undesirable for system performance.

Buffer pool overhead

DB2 incurs an overhead of about one page in database shared memory for every 30 buffer pool pages. This is a change from previous versions of DB2, where the overhead came out of the database heap (as specified by the **dbheap** database configuration parameter) and required resizing for very large buffer pools.

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.

Performance considerations

In general, the more memory that is made available for buffer pools without incurring operating system paging, the better the performance.

Large buffer pools provide the following advantages:

1. They 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 times and reducing the processor resource needed for I/O operations.
2. They provide the opportunity to achieve higher transaction rates with the same response time.
3. They reduce I/O contention for frequently used disk storage devices such as 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.

The following decisions need to be made regarding tuning buffer pools:

- Single or multiple buffer pools
- Size of the buffer pool
- Buffer pool assignment with multiple buffer pools
- Block-based buffer pools

- **`chngpgs_thresh`** and **`num_iocleaners`** considerations
- **`num_ioservers`** considerations

Each of these considerations are covered as follows:

Single or multiple buffer pools

With 32-bit versions of DB2, there is a limit on the total size of the buffer pools that can be defined regardless of the available real memory. No such restrictions apply to 64-bit versions of DB2 where real memory limits (causing operating system paging) are most likely the problem with defining large buffer pools.

The *advantages* of a single buffer pool are as follows:

- It exploits DB2's efficient page management algorithm to maintain a high buffer hit ratio by keeping the most active pages and important pages such as index pages in memory, while migrating less frequently used pages to disk.
- It requires no tuning once its size is chosen.

The main *disadvantage* of a single buffer pool is that you cannot discriminate in favor of table spaces requiring high priority access with potentially low activity access rates, when there are lower priority table spaces with higher activity rates using the same buffer pool.

Note: When table spaces have different page sizes, a separate buffer pool must be defined for each page size. Apportioning the total memory available for buffer pools between the various page sizes implicitly involves a degree of prioritization.

The advantages and disadvantages of multiple buffer pools are the opposite of those for a single buffer pool. Multiple buffer pools offer greater flexibility for prioritizing the I/O performance of different table spaces, but require constant monitoring and tuning to keep them performing optimally.

Size of the buffer pool

The size of the buffer pool is dependent upon the nature of the application workload and the size of the table spaces.

- Smaller buffer pools can be used for 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. Another example of tables that could have small buffer pools are those that are always appended to, such as audit trails and logs.

- Larger buffer pools are beneficial when they include small tables that have data accessed repeatedly and frequently. If this buffer pool is sized appropriately, the pages of small tables have a better chance of being found, contributing to a lower response time and a lower transaction cost.

Note: System paging generally has a greater negative impact on performance than database I/Os. Therefore, it is better to minimize system paging by defining smaller buffer pools, instead of defining larger buffer pools when there is insufficient real memory to back up the larger allocation.

Attention: It is very important to balance the size of buffer pools with sort heap storage allocation in real memory-constrained environments. For BI environments in particular, it is very desirable to allocate a large sort heap size to improve the performance of sorts, which may leave less real memory available for defining large buffer pools.

Buffer pool assignment with multiple buffer pools

The purpose of defining multiple buffer pools is to discriminate in favor of higher priority table spaces even when they do not necessarily have high activity against them. It is therefore critical to isolate table spaces in buffer pools in a way that their individual activity rates do not interfere or contend with each other.

Typically:

- Separate high activity, high priority table spaces from other high activity, high priority table spaces.
- Separate low activity, high priority table spaces from high activity, low priority table spaces.
- Frequently updated tables and indexes should be kept separate from tables and indexes that are frequently queried but infrequently updated.
- Indexes may be separated from data (this is only possible for DMS table spaces).
- Temporary table spaces should be separated from tables and indexes for workloads that are sort-intensive.
- Low activity table spaces can share the same buffer pool.

Block-based buffer pools

The optimal usage of a block-based buffer pool depends upon the number of pages reserved for the block (**NUMBLOCKPAGES**) area and the specified block size (**BLOCKSIZE**). The *block size* is the granularity at which the prefetchers (db2pfchr) 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 the **BLOCKSIZE,** or not an integral multiple of the **BLOCKSIZE** 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 prefetcher (***db2pfchr***) allows some wasted pages (up to 50%) in each buffer pool block, but if too much of a block would be wasted, the prefetcher (***db2pfchr***) 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. If there are multiple table spaces with different extent sizes associated with this buffer pool, choose a block size that matches the smallest extent size of all the table spaces associated with this buffer pool.

Attention: Block-based buffer pools are intended for sequential prefetching. If applications do not use sequential prefetching, the block area of the buffer pool is wasted.

chngpgs_thresh and num_iocleaners considerations

When the **chngpgs_thresh** threshold is triggered, *all* page cleaners are triggered at the same time.

For workloads that have high update activity, such as OLTP environments, the setting of this threshold can have a significant performance impact, as follows:

- A high threshold value can have a significant negative performance impact. A high threshold causes the page cleaners to be dormant for a long time while the number of dirty pages accumulate. When the threshold is finally triggered, the starting of all the page cleaners and the large number of accumulated dirty pages which would need to be flushed to disk, could result in a significant system impact due to the flood of buffer writes.

- If the threshold is set too low, then unnecessary buffer writes would cause disk contention as well as CPU utilization due to starting of the page cleaners, thereby impacting overall system performance.

In general, it is desirable to smooth out the execution of the page cleaners and writing of dirty pages to disk.

num_ioservers considerations

The I/O servers (prefetchers) are used on behalf of the database agents to perform prefetch I/O and asynchronous I/O by applications and utilities such as backup and restore. The prefetch manager assigns the prefetch requests to the prefetchers. There is one prefetch queue that is monitored by the prefetch manager and shared between all configured prefetchers.

A prefetch request is normally broken into smaller I/O requests, with the number of I/O requests being determined by number of containers in the table space.

To ensure prefetchers do not read data into the buffer pool and overwrite pages before they are accessed, the amount of prefetching is controlled by the size of the buffer pool, the prefetch size, and the access plan generated by the optimizer.

Note: DB2 limits the percent of a buffer pool that can contain prefetched pages to ensure that prefetching does not swamp the entire buffer pool.

A large number of prefetchers can result in excessive prefetching, and this may cause overwriting of pages in the buffer pool that were to be accessed by other applications. These overwritten pages will cause applications to suffer a buffer-hit miss, resulting in synchronous I/O. Excessive prefetching may also waste space in the buffer pool if the application does not process all the pages prefetched.

A small number of prefetchers can result in high prefetch times, thus causing the DB2 agents to perform synchronous I/O themselves.

Attention: 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

Best practices

We recommend the following best practices for tuning buffer pools and setting of configuration parameters **`chngpgs_thresh, num_iocleaners`** and **`num_ioservers`**:

1. For OLTP environments, typically allocate about 75% to 80% of the memory installed on the database server to buffer pools—assuming a dedicated database server.

   If not, allocate 75% to 80% of usable memory to DB2 (server memory configuration minus operating system and any other system requirements) to the buffer pool(s).

   As mentioned earlier, it is very important to balance the size of buffer pools with sort heap storage allocation in real memory-constrained environments.
2. For BI environments, allocate about 50% to the buffer pools and the rest for sorts. It is very desirable to allocate a large sort heap size to improve the performance of sorts, which may leave less real memory available for defining large buffer pools.

   Note: 64-bit DB2 has virtually no practical limit on the size of buffer pools.

3. Buffer pool hit ratios measure the efficiency of a buffer pool, and should be as high as possible for a given workload. It reflects how frequently the request for index or data pages is handled directly from the buffer pool instead of being accessed from disk.

   Buffer pool hit ratios may be computed for data plus indexes, data only, or indexes only, as follows:

   ```
   Buffer Pool Hit Ratio for data = (1 - ((buffer pool data physical
   reads)/(buffer pool data logical reads))) * 100
   ```

   ```
   Buffer Pool Hit Ratio for indexes = (1 - ((buffer pool index physical
   reads)/(buffer pool index logical reads))) * 100
   ```

   ```
   Buffer Pool Hit Ratio for data and indexes = (1 - ((buffer pool data
   physical reads + buffer pool index physical reads)/(buffer pool data
   logical reads + buffer pool index logical reads))) * 100
   ```

   In particular, index hit ratios should typically be higher than data hit ratios for both OLTP and BI workloads.

   Determining the optimal buffer pool size for a given workload is a trial and error exercise based on hit ratios and user response time measurements, as shown in Figure 3-31 on page 252.

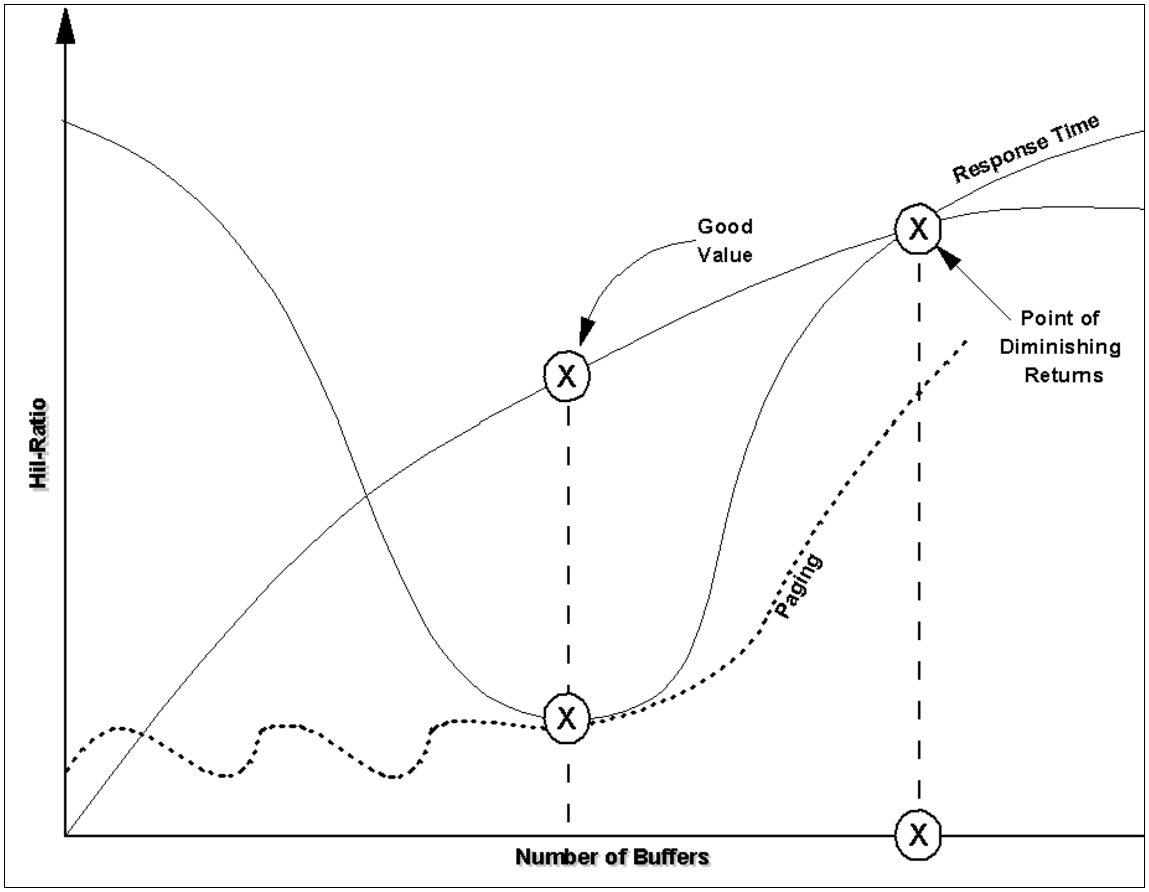


Figure 3-31 The number of buffers, hit ratios, response times, and paging

Important: Just computing the buffer hit ratio without taking into account the effect of system paging can lead to a spurious optimal value for the number of buffers. The actual optimal value is when the response times are best with low system paging, even though the hit ratios are not as high.

4. For OLTP environments:
   - Hit ratios above 95% are achievable, but anything above 80% is good.
   - Multiple buffer pools recommended:
     - Separate indexes and tables into different buffer pools (this requires the table spaces to be DMS table spaces).

        - Temporary table spaces should be assigned to a separate buffer pool to improve sort and global temporary table performance.
        - Journals and history tables where rows are appended to the end of the table can have their table spaces mapped to smaller buffer pools to reduce wasted memory.
        - Separate high activity, high priority table spaces from other high activity, high priority table spaces.
        - Separate low activity, high priority table spaces from high activity, low priority table spaces.
        - Frequently updated tables and indexes should be kept separate from tables and indexes that are frequently queried but infrequently updated.
        - Low activity table spaces can share the same buffer pool.
    - Extended storage is not appropriate for 64-bit DB2, and probably not appropriate if the system is CPU-bound.
    - Block-based buffer pools are not appropriate unless there is reasonable sequential prefetching, in which case a small percentage (between 10% and 20%) of the buffer pool pages (**NUMBLOCKPAGES**) should be reserved for prefetching, and the block size should be equal to the table space extent size. If there are multiple table spaces with different extent sizes associated with this buffer pool, choose a block size that matches the smallest extent size of all the table spaces associated with this buffer pool.
    - Reduce the default value of **chngpgs_thresh** to 20 - 30% to smooth out the writing of dirty pages to disk, and have dirty pages written out asynchronously due to the **chngpgs_thresh** threshold trigger, rather than LSN gaps or dirty pages stolen triggers.
    - Set **num_iocleaners** to be between one and the number of physical storage devices used for the database. Environments with high update transaction rates and a large number of buffer pools require more page cleaners to be configured. The number of page cleaners should generally not exceed the number of CPUs in the system.
    - Let **num_ioservers** default, but monitor and adjust as necessary.
- For BI environments:
    - The ratio of asynchronous reads-to-synchronous reads is much more important in BI environments than the hit ratio because of the large volumes of data being processed resulting in poor hit ratios. The objective is to strive for almost 100% asynchronous reads and minimal synchronous reads by the DB2 agents.
    - Multiple buffer pools recommended for indexes and temporary table spaces, but not for data:

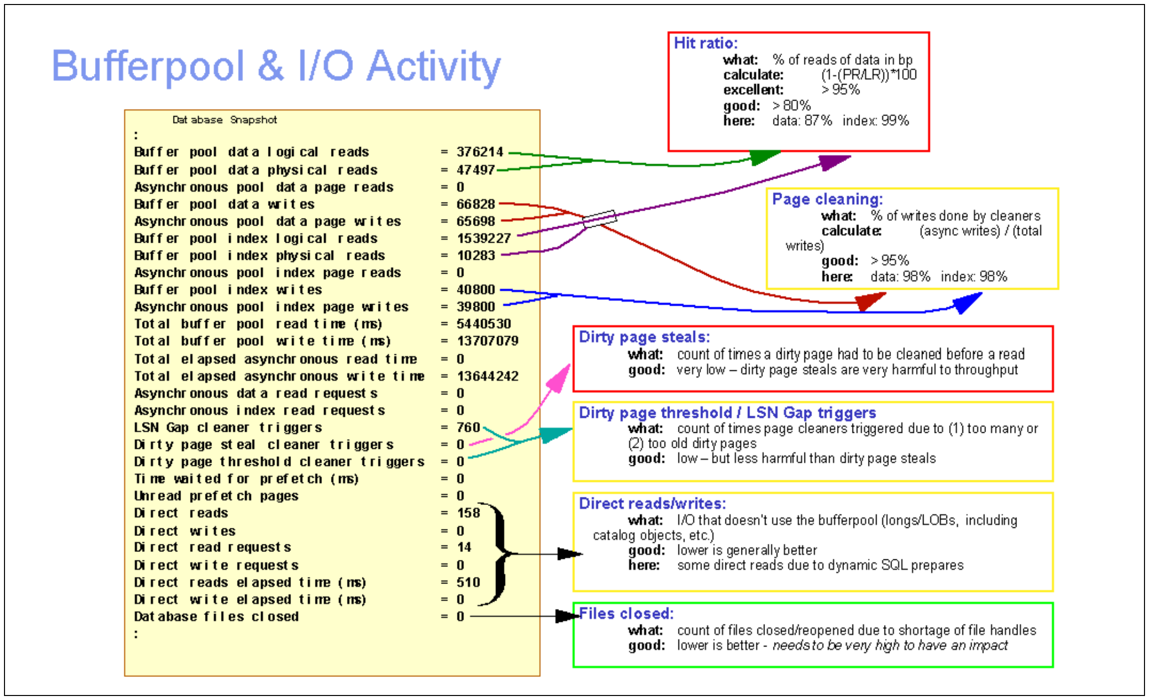
  - Separate indexes and tables into different buffer pools (this requires the table spaces to be DMS table spaces).
  - Temporary table spaces should be assigned to a separate buffer pool to improve sort-intensive queries of such environments.
  - Combine table spaces with the same page size into one buffer pool for optimal data hit ratios.
- – Extended storage may be appropriate since BI environments tend to be I/O-bound (unless 64-bit version DB2 is used).
- – Block-based buffer pools are highly recommended since BI environments tend to have significant prefetching activity. A high percentage (80% or more) of pages in the buffer pool can be reserved for prefetching (**NUMBLOCKPAGES**), and the block size should be equal to the extent size of the table space. If there are multiple table spaces with different extent sizes associated with this buffer pool, choose a block size that matches the smallest extent size of all the table spaces associated with this buffer pool.
- – Let **chngpgs_thresh** default to 60% since BI environments are predominantly read only and accumulated dirty page writes should not be a problem.
- – Let **num_iocleaners** default. The number of page cleaners should generally not exceed the number of CPUs in the system.
- – Increase **num_ioservers** to the number of disks in the system for maximum I/O parallelism; if there are too many disks, use a trial and error mechanism to determine the optimal number.

Performance monitoring metrics

Metrics to monitor may be obtained from the snapshot monitor, as shown in Figure 3-32 on page 255 and Figure 3-33 on page 255 and Figure 3-34 on page 256.

Select fields are shown from the results of the **get snapshot for bufferpools on <database_alias>** command in each case.

- ► Figure 3-32 on page 255 is used to monitor the buffer pool hit ratios and page cleaning considerations.
- ► Figure 3-33 on page 255 is used to monitor the efficacy of block-based buffer pools.
- ► Figure 3-34 on page 256 is used to monitor the efficacy of prefetching.



Bufferpool & I/O Activity

```
        Database Snapshot
:
Buffer pool data logical reads              = 376214
Buffer pool data physical reads             = 47497
Asynchronous pool data page reads           = 0
Buffer pool data writes                     = 66828
Asynchronous pool data page writes          = 65698
Buffer pool index logical reads             = 1539227
Buffer pool index physical reads            = 10283
Asynchronous pool index page reads          = 0
Buffer pool index writes                    = 40800
Asynchronous pool index page writes         = 39800
Total buffer pool read time (ms)            = 5440530
Total buffer pool write time (ms)           = 13707079
Total elapsed asynchronous read time        = 0
Total elapsed asynchronous write time       = 13644242
Asynchronous data read requests             = 0
Asynchronous index read requests            = 0
LSN Gap cleaner triggers                    = 760
Dirty page steal cleaner triggers           = 0
Dirty page threshold cleaner triggers       = 0
Time waited for prefetch (ms)               = 0
Unread prefetch pages                       = 0
Direct reads                                = 158
Direct writes                               = 0
Direct read requests                        = 14
Direct write requests                       = 0
Direct reads elapsed time (ms)              = 510
Direct write elapsed time (ms)              = 0
Database files closed                       = 0
:
```

Hit ratio:
what: % of reads of data in bp
calculate: (1-(PR/LR))*100
excellent: > 95%
good: > 80%
here: data: 87% index: 99%

Page cleaning:
what: % of writes done by cleaners
calculate: (async writes) / (total writes)
good: > 95%
here: data: 98% index: 98%

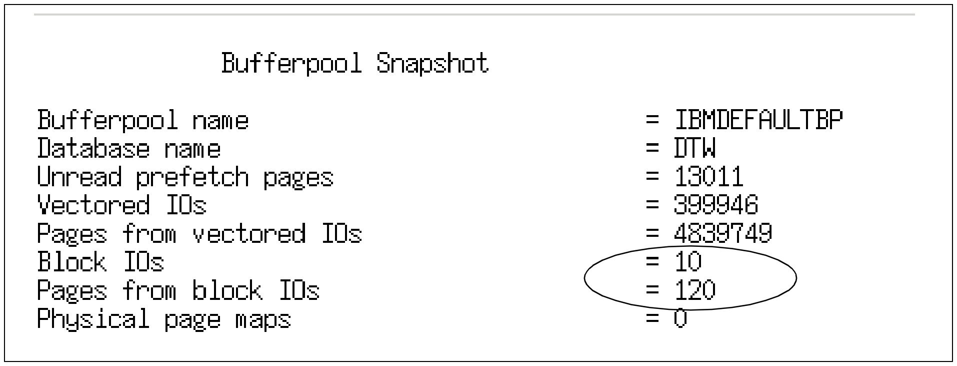
Dirty page steals:
what: count of times a dirty page had to be cleaned before a read
good: very low – dirty page steals are very harmful to throughput

Dirty page threshold / LSN Gap triggers
what: count of times page cleaners triggered due to (1) too many or (2) too old dirty pages
good: low – but less harmful than dirty page steals

Direct reads/writes:
what: I/O that doesn't use the bufferpool (longs/LOBs, including catalog objects, etc.)
good: lower is generally better
here: some direct reads due to dynamic SQL prepares

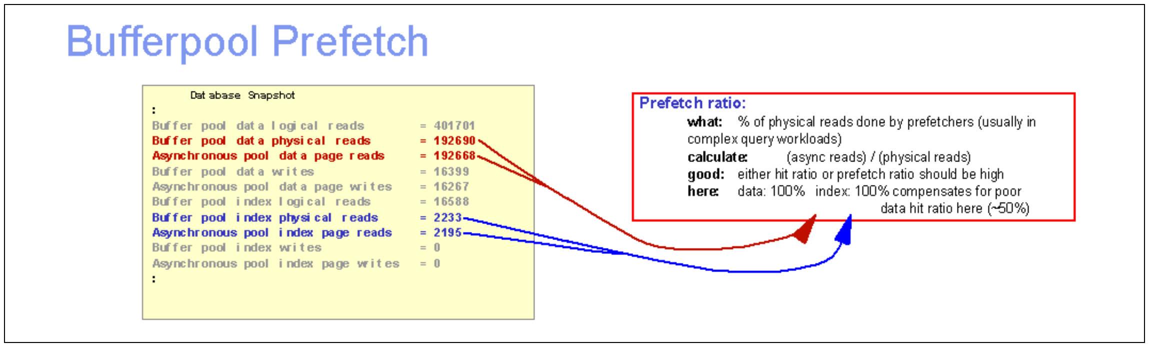
Files closed:
what: count of files closed/reopened due to shortage of file handles
good: lower is better - *needs to be very high to have an impact*

Figure 3-32 Output of buffer pool snapshot on the database and calculation



```
              Bufferpool Snapshot

Bufferpool name                       = IBMDEFAULTBP
Database name                         = DTW
Unread prefetch pages                 = 13011
Vectored IOs                          = 399946
Pages from vectored IOs               = 4839749
Block IOs                             = 10
Pages from block IOs                  = 120
Physical page maps                    = 0
```

Figure 3-33 Buffer pool snapshot showing Block I/O value



Bufferpool Prefetch

```
        Database  Snapshot
:
Buffer pool data logical reads            = 401701
Buffer pool data physical reads           = 192690
Asynchronous pool data page reads         = 192668
Buffer pool data writes                   = 16399
Asynchronous pool data page writes        = 16267
Buffer pool index logical reads           = 16588
Buffer pool index physical reads          = 2233
Asynchronous pool index page reads        = 2195
Buffer pool index writes                  = 0
Asynchronous pool index page writes       = 0
:
```

Prefetch ratio:

what: % of physical reads done by prefetchers (usually in complex query workloads)

calculate: (async reads) / (physical reads)

good: either hit ratio or prefetch ratio should be high

here: data: 100% index: 100% compensates for poor data hit ratio here (~50%)

Figure 3-34 Database snapshot showing prefetch details and calculation

> **Important:** All the fields in the snapshot monitor, whether they are water marks, counters or gauges, should be monitored over many representative intervals spread over a few weeks, to detect consistent trends before reacting with configuration changes.
>
> Note that counter type monitoring elements should be reset at the beginning of each monitoring interval by issuing the **`RESET MONITOR`** command.

The fields of interest in all the figures for tuning the buffer pool, **`chngpgs_thresh`**, **`num_iocleaners`**, **`num_ioservers`** are as follows:

- `Buffer pool data logical reads` is a counter that records the number of logical read requests for data pages that have gone through the buffer pool.
- `Buffer pool data physical reads` is a counter that records the number of read requests that required I/O to get data pages into the buffer pool.
- `Asynchronous pool data page reads` is a counter that records the number of asynchronous read data page requests.
- `Buffer pool data writes` is a counter that indicates the number of times a buffer pool data page was physically written to disk.
- `Asynchronous pool data page writes` is a counter that records the number of times a buffer pool data page was physically written to disk by either an asynchronous page cleaner, or a prefetcher. A prefetcher may have written dirty pages to disk to make space for the pages being prefetched.
- `Buffer pool index logical reads` is a counter that records the number of logical read requests for index pages that have gone through the buffer pool.
- `Buffer pool index physical reads` is a counter that records the number of read requests that required I/O to get index pages into the buffer pool.

- `Asynchronous pool index page reads` is a counter that records the number of asynchronous read index page requests.
- `Buffer pool index writes` is a counter that indicates the number of times a buffer pool index page was physically written to disk.
- `Asynchronous pool index page writes` is a counter that records the number of times a buffer index data page was physically written to disk by either an asynchronous page cleaner, or a prefetcher. A prefetcher may have written dirty pages to disk to make space for the pages being prefetched.
- `Total buffer pool read time (ms)` is a counter that records the total amount of elapsed time spent processing read requests that caused data or index pages to be physically read from disk to buffer pool.
- `Total elapsed asynchronous read time` is a counter that records the total elapsed time spent reading by database manager prefetchers.
- `LSN Gap cleaner triggers` is a counter that records the number of times a page cleaner was invoked because the logging space used had reached a predefined criterion for the database.
- `Dirty page steal cleaner triggers` is a counter that records the number of times a page cleaner was invoked because a synchronous write was needed during the victim buffer replacement for the database.
- `Dirty page threshold cleaner triggers` is a counter that records the number of times a page cleaner was invoked because a buffer pool had reached the dirty page threshold (**`chngpgs_thresh`**) criterion for the database.
- `Time waited for prefetch (ms)` is a counter that records the total amount of time an application waited for a prefetcher to finish loading pages into the buffer pool.
- `Unread prefetch pages` is a counter that records the number of pages that the prefetcher read in that were never used.
- `Direct reads` is a counter that records the number of read operations that do not use the buffer pool.
- `Direct writes` is a counter that records the number of write operations that do not use the buffer pool.
- `Direct read requests` is a counter that records the number of requests to perform a direct read of one or more sectors of data.
- `Direct write requests` is a counter that records the number of requests to perform a direct write of one or more sectors of data.
- `Direct reads elapsed time (ms)` is a counter that records elapsed time (in milliseconds) required to perform the direct reads.

- `Direct writes elapsed time (ms)` is a counter that records elapsed time (in milliseconds) required to perform the direct writes.
- `Database files closed` is a counter that records the total number of database files closed.
- `Vectored IOs` is a counter that records the number of vectored I/O requests.
- `Pages from vectored IOs` is a counter that records the total number of pages read by vectored I/O.
- `Block IOs` is a counter that records the number of block I/O requests.
- `Pages from block IOs` is a counter that records the total number of pages read by block I/O.
- `Last reset timestamp` (not shown in any of the figures) is a timestamp element type and indicates the date and time that the monitor counters were reset for the application issuing the **`get snapshot`** command.
- `Snapshot timestamp` (not shown in any of the figures) is a timestamp when the **`get snapshot`** command was issued.

Compute the following values:

Prefetch ratio as follows:

```
Prefetch Ratio = (((Asynchronous pool data page reads) + (Asynchronous pool
index page reads)) / ((Buffer pool data logical reads) + (Buffer pool index
logical reads))) * 100
```

Percentage of dirty page steal cleaner triggers as follows:

```
Percentage of dirty page steal cleaner triggers = ((Dirty page steal cleaner
triggers) / ((Dirty page steal cleaner triggers) + (Dirty page threshold
cleaner triggers) + (LSN Gap cleaner triggers))) * 100
```

Consider tuning the various parameters under the following circumstances:

- For OLTP and BI environments, if the buffer hit ratios are less than 80%, then consider adding buffers in increments to try to improve the Buffer Pool Hit Ratio for data and indexes, Buffer Pool Hit Ratio for data and Buffer Pool Hit Ratio for indexes as defined earlier. Aim for higher Buffer Pool Hit Ratio for indexes ratios.
- Prefetching performance can be analyzed by comparing the amount of synchronous versus asynchronous I/O as computed by Prefetch Ratio above.

  If the Prefetch Ratio is very small, then there is little prefetch activity happening. This may be due to less **`num_ioservers`** configured, or a workload reading very few rows at a time thereby inhibiting prefetching, or all the table spaces in the database are set up with only one container each so that prefetching normally does not occur.

Note: When `DB2_PARALLEL_IO` is enabled, prefetching can occur within a single container table space if the prefetch size is a multiple of the extent size.

Another consideration is the Time waited for prefetch (ms); a high value may indicate an insufficient number of prefetchers since applications are waiting for the prefetch to complete. Other causes of a high value may be an poorly tuned I/O subsystem resulting in elongated I/I times.

- A high value in the Unread prefetch pages counter indicates that prefetchers are causing unnecessary I/O by reading pages into the buffer pool that will not be used. This may indicate a very large prefetch size that could perhaps be reduced.
- If the Percentage of dirty page steal cleaner triggers is high, it indicates that page cleaners are being triggered too aggressively which defeats one purpose of the buffer pool that is to defer writing to the last possible moment. A high value indicates that dirty pages were written out synchronously to disk, and that the application had to wait for these I/Os to complete. This requires re-evaluating the values set for **`chngpgs_thresh`** (lower the percentage), **`softmax`** (reduce the percentage) and **`num_iocleaners`** (increase the number). The objective is to flush dirty pages more frequently to avoid the synchronous flushing of dirty pages. This value should be very low.
- For block-based buffer pools, the ratio (Pages from block IOs/Block IOs) provides the number of pages read per block I/O. If this value is much less than the **`BLOCKSIZE`** defined for the block-based buffer pool, than it indicates that the **`BLOCKSIZE`** is too large and could be made smaller.

Note: DB2 allows wasted space in a block during a block I/O because it is possible that some pages were already found in the buffer pool. Currently, DB2 can waste up to 50% of a block. Anything more than that will cause DB2 to do page-based I/O into the page area of the buffer pool.

- If the number of Vectored IOs is significantly larger than the number of Block IOs, it may indicate that the buffer pool is poorly configured for block-based I/O and **`NUMBLOCKPAGES`** may need to be increased.
- Consider the following for the **`num_cleaners`** parameter:
  - Reduce **`num_iocleaners`** if:

  - Buffer pool data writes is approximately equal to Asynchronous pool data page writes
  - Buffer pool index writes is approximately equal to Asynchronous pool index writes
- Increase **`num_iocleaners`** if:
  - Buffer pool data writes is much greater than Asynchronous pool data page writes
  - Buffer pool index writes is much greater than Asynchronous pool index page writes

3.4.6 Locking considerations

Concurrency is critical to the performance and throughput of multi-user environments accessing shared data. Throughput and concurrency depends upon application design and database configuration parameters. Application design considerations are discussed in 3.3.7, "Concurrency" on page 186; however, we cover the runtime considerations here.

The size of a lock is depends upon the mode of DB2, and whether it is the first lock acquired on the database object, as shown in Table 3-12.

Table 3-12 Lock space usage details

DB2 bit mode	Bytes per lock	Condition
DB2 32-bit mode	72 bytes	First lock on an object
	36 bytes	Subsequent locks on an object
DB2 64-bit mode	112 bytes	First lock on an object
	56 bytes	Subsequent locks on an object

An important concept that can have a significant impact on concurrency is *lock escalation*, which is the process of replacing large number of row locks with a single table lock. Escalation occurs from row locks to a table lock when the number of locks held exceed the thresholds defined by the database configuration parameters **`locklist`** and **`maxlocks`**. A table can be escalated in share or exclusive mode, with exclusive mode being significantly more detrimental to concurrency and throughput. Lock escalations can result in reduced concurrency, lock waits, and deadlocks.

Figure 3-35 on page 261 describes the lock escalation process when **`maxlocks`** is exceeded.

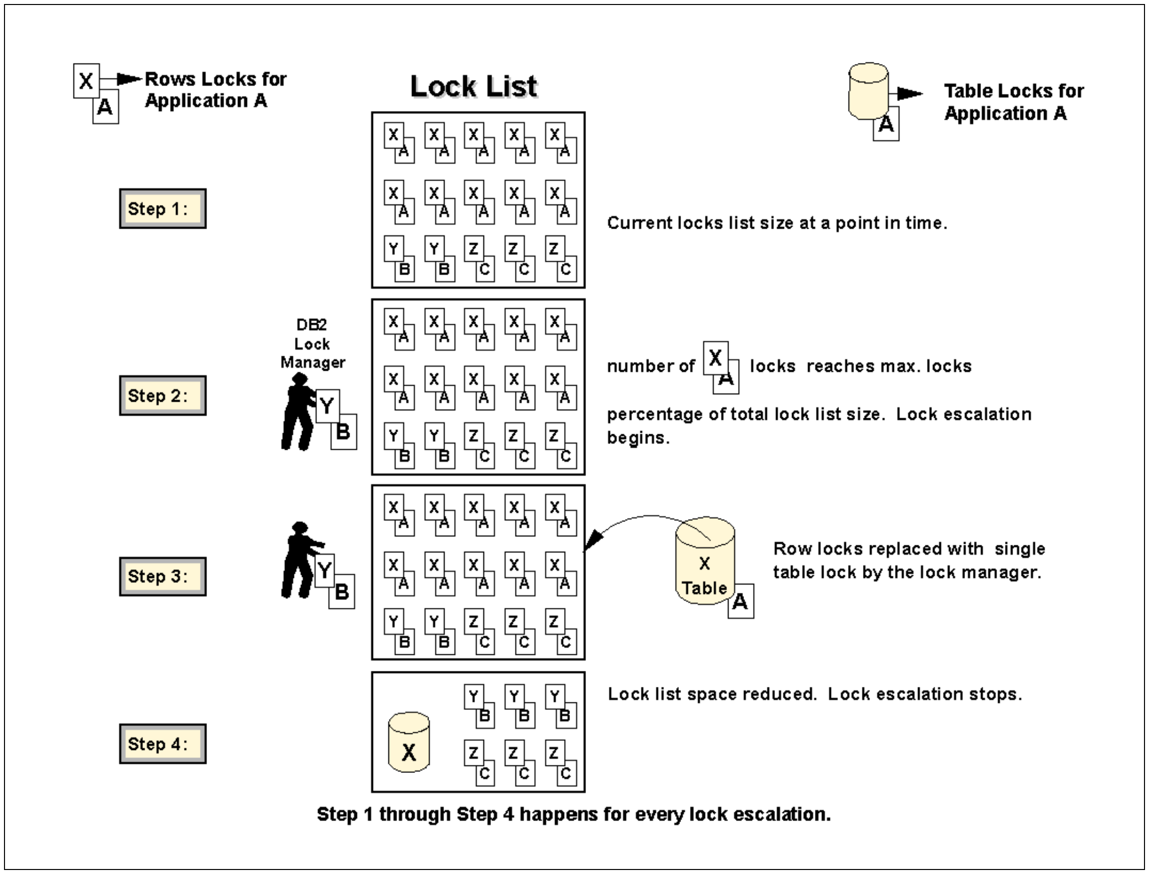

Rows Locks for Application A
Lock List
Table Locks for Application A
Step 1:
Current locks list size at a point in time.
DB2 Lock Manager
Step 2:
number of X A locks reaches max. locks percentage of total lock list size. Lock escalation begins.
Step 3:
X Table A
Row locks replaced with single table lock by the lock manager.
Step 4:
Lock list space reduced. Lock escalation stops.
Step 1 through Step 4 happens for every lock escalation.

Figure 3-35 Process of lock escalation

In step 1, applications A, B, and C take row locks on tables X, Y, and Z respectively, which occupy storage in the lock list.

In step 2, locks taken by application A causes the `maxlocks` threshold to be exceeded which triggers the lock escalation process.

In step 3, all the rows locks held by application A on table X are replaced by a single table lock. If there had been multiple tables accessed by the application, then it would have considered the table with the most row locks first as the candidate for lock escalation.

In step 4, lock escalation stops when lock list utilization by this application drops below the maxlocks level.

The following database configuration parameters affect runtime concurrency and throughput:

- **`locklist`**
- **`maxlocks`**
- **`locktimeout`**
- **`dlchktime`**

Each of these parameters are reviewed in turn, with a brief description of their function, performance considerations associated with their use, best practices for using them, and monitoring metrics for evaluating their effectiveness.

locklist

This parameter specifies the maximum amount of storage available to store all the locks concurrently held within a database. This storage is allocated when the first application connects to the database, and is freed when the last application disconnects from the database.

When lock escalation occurs as a result of a lock list full condition, it might not affect the table that acquires the lock that triggers escalation. Lock escalation continues with tables with the most row locks until lock list utilization falls to about half of the total **`locklist`**. The application receives a -912 sqlcode when the maximum number of locks requests have been reached for the database.

The default value varies by platform and DB2 32-bit or 64-bit mode as follows:

- For UNIX
  - 100 4KB pages
- For Windows
  - Database server with local and remote clients 50 4KB pages
  - 32-bit Database server with local clients 25 4KB pages
  - 64-bit Database server with local clients 50 4KB pages

This parameter can be changed online—but can only be increased online. To decrease the value of **`locklist`**, the database has to be reactivated after making the change.

Performance considerations

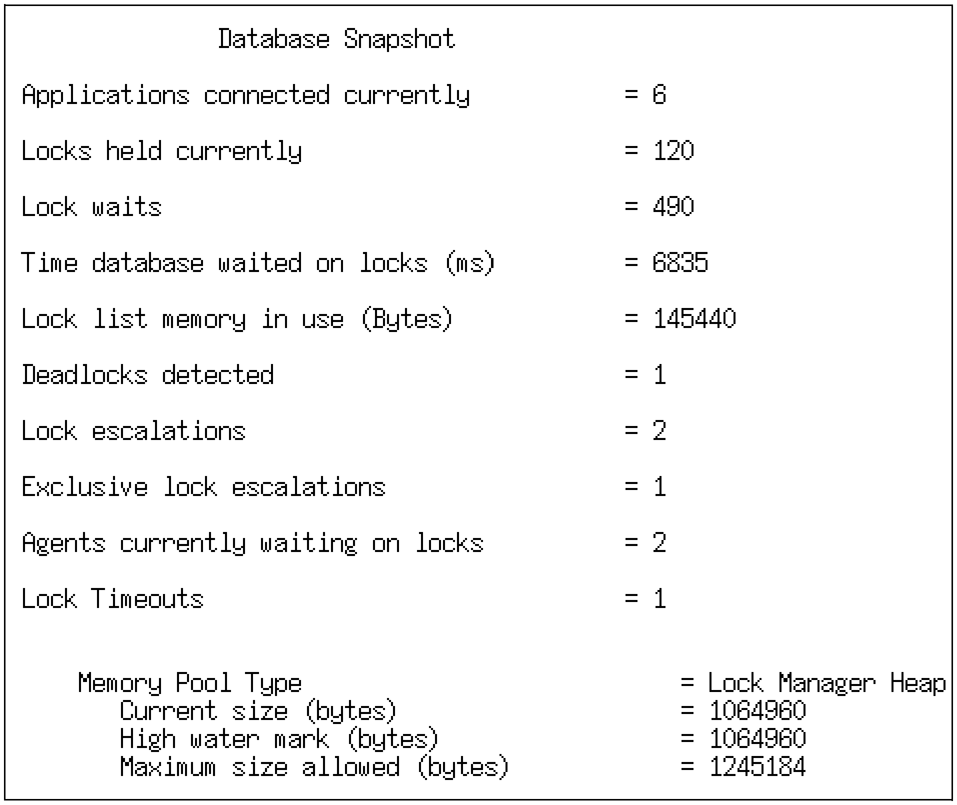
An underconfigured value for this parameter can result in lock escalations which can significantly degrade performance, while an overconfigured **`locklist`** can waste valuable memory that could be better utilized in other heaps.

Best practices

The lock list should accommodate the total number of concurrent locks held by all concurrent applications.

We recommend the following best practices for tuning `locklist`:

- For an existing system, the **`get snapshot for database`** command as shown in Figure 3-36 provides assistance in sizing locklist as follows:



```
              Database Snapshot

Applications connected currently          = 6

Locks held currently                      = 120

Lock waits                                = 490

Time database waited on locks (ms)        = 6835

Lock list memory in use (Bytes)           = 145440

Deadlocks detected                        = 1

Lock escalations                          = 2

Exclusive lock escalations                = 1

Agents currently waiting on locks         = 2

Lock Timeouts                             = 1


    Memory Pool Type                          = Lock Manager Heap
       Current size (bytes)                   = 1064960
       High water mark (bytes)                = 1064960
       Maximum size allowed (bytes)           = 1245184
```

Figure 3-36   Database snapshot for lock information

Important: All the fields in the snapshot monitor, whether they are water marks, counters or gauges, should be monitored over many representative intervals spread over a few weeks, to detect consistent trends before reacting with configuration changes.

Note that counter type monitoring elements should be reset at the beginning of each monitoring interval by issuing the **`RESET MONITOR`** command.

Compute the average number of locks per application (ANLA) as:

```
ANLA = (Locks held currently / Applications connected currently)
```

Choose a locklist value between the following lower and upper limits:

– Lower limit can be computed as:

```
((ANLA * (36 or 56 bytes) * maxappls) / 4096)
```

– Upper limit can be computed as:

```
((ANLA * (72 or 112 bytes) * maxappls) / 4096)
```

Note: If **`maxappls`** is set to **`AUTOMATIC`**, consider using the highest value reported in the `Application connected currently` field over many representative intervals spread over a few weeks, to detect consistent trends before reacting with configuration changes.

Note that counter-type monitoring elements should be reset at the beginning of each monitoring interval by issuing the **`RESET MONITOR`** command.

- For new systems, the same formula should be used with an estimate of the number of concurrent locks
  - OLTP transactions typically hold a maximum of less than 20 concurrent locks
  - BI queries, the number of concurrent locks held per query should be much lower than OLTP transactions, given the read only nature of such workloads.
- For OLTP & BI environments, lock escalations should be avoided completely or kept to an absolute minimum to avoid significant performance degradation.

Performance monitoring metrics

Metrics to monitor lock-related information may be obtained from the Snapshot Monitor as well as application concurrency health indicators in the Health Center, as shown in Figure 3-26 on page 222.

Figure 3-36 on page 263 lists selected fields from the **`get snapshot for db`** command.

Important: All the fields in the snapshot monitor, whether they are water marks, counters or gauges, should be monitored over many representative intervals spread over a few weeks, to detect consistent trends before reacting with configuration changes.

Note that counter-type monitoring elements should be reset at the beginning of each monitoring interval by issuing the **`RESET MONITOR`** command.

The following fields are of interest for tuning **`locklist:`**

- `Lock waits` is a counter that records the number of times that applications or connections waited for locks.
- `Time database waited on locks (ms)` is a counter that records the total elapsed time waited for a lock.
- `Deadlocks detected` is a counter that records the total number of deadlocks that have occurred.
- `Lock escalations` is a counter that records the total number of times that locks have been escalated from several rows to a table lock.
- `Exclusive lock escalations` is a counter that record the total number of times that locks have been escalated from several rows to a table lock.
- `Lock timeouts` is a counter that records the number of times that a request to lock an object timed-out instead of being granted.

The average lock wait time (ALWT) can be computed as follows:

```
   ALWT = (Time database waited on locks (ms) / Lock waits)
```

Figure 3-26 on page 222 shows the Health Center lock escalation health indicators such as lock escalation rate and lock list utilization, which can also be used to alert you to potential concurrency problems.

Note: The `Lock list memory in use (Bytes)` field is a gauge rather than a water mark, and therefore not appropriate for sizing **`locklist`**.

If the `Lock escalations` field is non-zero and `Lock waits`, average lock wait time (ALWT)**,** `Deadlocks detected`, and `Lock timeouts` are high, then lock escalations should be eliminated. Since there is no definitive way to determine the precise cause of lock escalations[10] (**`locklist`** full or **`maxlocks`** threshold exceeded for an

[10] lock list full returns a -912 sqlcode to the application; however, the occurrence of such application errors may not be discernible to the DBA.

application), a combination of the following approaches must be considered to avoid lock escalations:

1. Increase **`locklist`** and use a trial and error method to determine the optimal value.
2. Increase the **`maxlocks`** threshold, as discussed in “maxlocks” on page 266.
3. Have those applications identified through the administration notification log messages (**`notifylevel`** 4) as shown in Figure 3-24 on page 217 use less restrictive isolation levels (CS or UR) and issue more frequent commits to reduce the number of concurrent locks (however, this is unlikely to be a timely resolution to an immediate problem).

Attention: The new High water mark (bytes) value of Lock Manager Heap in the **`get snapshot for dbm | database`** command should *not* be used to set the **`locklist`** parameter. Since the locklist is allocated in full when the first application connects to the database, this high water mark always reflects this initial allocation. Treat this watermark as a placeholder for a future release enhancement.

maxlocks

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 some of the locks held by that application.

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, then lock escalation will stop. Otherwise, it will continue until the percentage of the lock list is below the value of **`maxlocks`**.

The **`maxlocks`** parameter multiplied by the **`maxappls`** parameter cannot be less than 100.

The default value is 10% for UNIX and 22% for Windows.

This parameter is configurable online.

Performance considerations

`maxlocks` and **`locklist`** are interdependent parameters, and **`maxlocks`** can have a significant negative impact on concurrency and throughput.

If **maxlocks** is underconfigured, then lock escalations happen when there is still enough lock space for other concurrent applications in **locklist**.

If **maxlocks** is overconfigured, then a few applications can consume most of the **locklist** and other applications will have to perform lock escalation due to lock space full condition.

Best practices

We recommend the following best practices for tuning **maxlocks**:

1. For OLTP environments, choose a value between 10% and 20%, since transactions typically hold very few concurrent locks and there are a large number of concurrent users.
2. For a large number of concurrent applications, consider:

   ```
   maxlocks ((2*100)/ maxappls)
   ```

   Where 2 is used to achieve twice the average, and 100 represents the largest percentage value allowed.
3. For a few applications that run concurrently, consider:

   ```
   maxlocks= ((2*100)/ Applications connected currently)
   ```
4. For BI environments, consider reducing the **maxlocks** parameters to below that of OLTP, since queries are primarily read only and rarely hold many concurrent locks.

Performance monitoring metrics

All the monitoring metrics discussed for **locklist** apply to **maxlocks**, as well.

With an Event Monitor, you can determine the maximum number of locks (**locks_held_top** monitoring element) held for an individual application and use it to tune the value of **maxlocks**. For details on creating an Event Monitor with transaction events, refer to *DB2 UDB System Monitor Guide and Reference,* SC09-4847.

locktimeout

This parameter specifies the number of seconds that an application will wait to obtain a lock.

The default is -1, which means that lock timeout detection is turned off; this means that the application waits until the lock is granted or a deadlock occurs and it is chosen as the victim.

Performance considerations

If **locktimeout** is set very high, then an application's run time may increase due to potentially long lock waits and the possible occurrence of deadlocks. If

locktimeout is set too low, spurious timeouts occur when the application could have completed successfully, had they waited a little bit longer for the lock to be granted.

When an application times out, it receives an error message SQL0911N with a reason code of 68, and allows other applications to proceed. The timed-out application may choose to retry the request or terminate with an error message.

Best practices

We recommend the following best practices for tuning **locktimeout**:

1. For OLTP environments with short-lived transactions, set **locktimeout** to a few seconds (less than 10 seconds), and then tune it to an optimal value.

   Do *not* let it default to -1.

2. For BI environments which tend to run much longer, set **locktimeout** to a higher value (perhaps 60 seconds), and then tune it to an optimal value.

Performance monitoring metrics

Figure 3-36 on page 263 provides information about lock waits in the fields `Lock Timeouts` and average wait time for a lock (`Time database waited on locks (ms) / Lock waits`).

Consider the following:

- Increase the **locktimeout** value when there are too many lock timeouts.
- Decrease the **locktimeout** value if the average lock wait times are too high.

Additional details may be obtained in the administration notification log when the **notifylevel** is set to 4.

Note: **notifylevel** 4 should only be used for problem determination in short bursts, because it incurs overhead.

For more details on the administration notification log, refer to *DB2 UDB Administration Guide: Performance*, SC09-4821.

dlchktime

This parameter specifies the frequency at which the database manager checks for deadlocks among all the applications connected to the database.

The default is 10000 milliseconds.

Performance considerations

A high value for this parameter decreases the frequency of checking for deadlocks, and can cause applications to wait longer than necessary for the resolution of a deadlock condition. A low value for this parameter increases the frequency of checking for deadlocks and can therefore decrease run-time performance due to increased database manager deadlock checking.

Best practices

We recommend that you let **`dlchktime`** default to 10 seconds for both OLTP and BI environments.

Performance monitoring metrics

In UNIX, the db2dlock process performs deadlock detection. Monitor this process for high CPU utilization using the **`ps aux | grep db2dlock`** operating system command and increase the value of **`dlchktime`** if the CPU consumption of this process is more than 1% or 2% in a CPU-constrained environment.

In Windows, this function is performed as a thread under the db2sysc process and cannot be monitored.

3.4.7 Package cache considerations

The **`pckcachesz`** database configuration parameter specifies the size of the cache used for caching sections for static and dynamic SQL statements in the database. This is allocated out of database shared memory, and is allocated when the database is initialized and freed when the database is shut down.

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. Performance is enhanced when the same package or dynamic statement is used multiple times by applications connected to the database.

Sections for static and dynamic SQL statements are kept in the package cache until one of the following happens:

- The database is shut down.
- The static package or the dynamic SQL is invalidated.
- The package cache runs out of space.
- The package cache is reset by issuing the **`FLUSH PACKAGE CACHE`** command.

When a new package needs to be retrieved from disk and inserted into the package cache, but there is no more space in the cache and all the packages in the cache are in use, then overflows occur to other database memory heaps.

However, if there is no unused space in the package cache, but there are unused packages in the package cache, then a victim is chosen from the unused packages and its space is used for the incoming package.

The default value is -1, which corresponds to the minimum of 32 4 K pages, or 8 times the value of **`maxappls.`** This is a soft limit and can overflow to other database shared memory heaps such as **`dbheap`**, **`util_heap_sz`**, and **`catalogcache_sz`**. This cache can also have spill-ins from **`catalogcache_sz.`**

This parameter is configurable online.

Performance consideration

If this package cache is too small, performance can degrade due to the need for package access from disk or the need for compiling dynamic SQL statements. In addition, package cache overflows to other database shared memory heaps can impact catalog cache hit ratio and result in overall system performance degradation.

If the package cache is too big, then memory would be wasted holding copies of the initial sections; this memory would be better served used by other database shared memory heaps such as buffer pools or the catalog cache.

Best practices

We recommend the following best practices for tuning **`pckcachesz`**. For both OLTP and BI environments:

- Start with the default value, and tune its value using the Database System Monitor.
- Strive for a high package cache hit ratio of 0.8 or more, and eliminate overflows to other heaps.

  Attention: For OLTP environments where a set of SQL statements are executed repeatedly, inserts into the package cache may be a more significant indicator of poor performance than overflows, since they incur constant recompiling of dynamic SQL statements that is expensive.

  In many cases where dynamic SQL is used without parameter markers, package cache hit ratios can be well below 0.8. In such cases, if the SQL cannot be changed, concentrate on tuning the catalog cache.
- Choose **`pckcachesz`** equal to (`Package cache high water mark / 4096`) from Figure 3-37 on page 271.

Performance monitoring metrics

Metrics to monitor lock-related information may be obtained from the snapshot monitor as well as package cache indicators in the Health Center as shown in Figure 3-26 on page 222.

Figure 3-37 lists selected fields from the **`get snapshot for db`** command.

```
db2 get snapshot for database on DTW | grep -i "Package cache"

Package cache lookups                        = 68054
Package cache inserts                        = 37
Package cache overflows                      = 0
Package cache high water mark (Bytes)        = 313530
```

Figure 3-37 Database snapshot - package cache monitor elements

Important: All the fields in the snapshot monitor, whether they are water marks, counters or gauges, should be monitored over many representative intervals spread over a few weeks to detect consistent trends before reacting with configuration changes.

Note that counter-type monitoring elements should be reset at the beginning of each monitoring interval by issuing the **`RESET MONITOR`** command.

All the fields in Figure 3-37 are of interest in tuning **`pckcachesz:`**

- `Package cache lookups` is a counter that records the number of times that an application looked for a section or package in the package cache. This indicates the overall number of references at the database level since the database was started or monitor data was reset. This is used to compute the package cache hit ratio.
- `Package cache inserts` is a counter that records the number of times that a requested section was not available for use and had to be loaded into the package cache. This is used to compute the package cache hit ratio.
- `Package cache overflows` is a counter that records the total number of times the package cache overflowed the bounds of its allocated memory.
- `Package cache high water mark` is a water mark that records the largest size (in bytes) reached by package cache (in bytes). If the package cache overflowed, then this element contains the largest size reached by the package cache during overflow.

Compute the package cache hit ratio (PCHR) as follows:

```
PCHR = (1 - (Package cache inserts / Package cache lookups))
```

Consider *increasing* the value of **pckcachesz** under the following circumstances:

- The package cache hit ratio (PCHR) is below 0.8; use a trial and error method of increasing the size of the package cache until PCHR stops increasing.
- Package cache overflows is greater than zero.
- For OLTP environments, if there is significant Package cache inserts in a steady state even without Package cache overflows, since this implies constant recompiling of dynamic SQL statements.

Consider *decreasing* the value of **pckcachesz** when ((`Package cache high water mark) / 4096`) is consistently less than the value of **pckcachesz.**

3.4.8 Catalog cache considerations

The **catalogcache_sz** database configuration parameter specifies the size of the cache that is allocated out of database shared memory. This memory gets allocated when the database is initialized and freed when the database is shut down. The catalog cache stores the following:

- **SYSTABLES** information, including package descriptors
- Authorization information, including **SYSDBAUTH** information and execute privileges for routines
- **SYSROUTINES** information

The use of the catalog cache can help improve the performance of the following if most of the required information can be found in the cache:

- Binding packages and compiling SQL statements
- Operations involving checking of database level privileges
- Operations involving checking of execute privileges for routines such as stored procedures and UDFs

When a new object needs to be retrieved from disk and inserted into the catalog cache, but there is no more space in the cache and all the objects in the cache are in use, then object overflows occur to other database memory heaps. However, if there is no unused space in the catalog cache, but there are unused objects in the catalog cache, then a victim is chosen from the unused objects and its space is used for the incoming object.

The default value is -1 corresponds to the minimum of 8 4 K pages, or 4 times the value of **`maxappls.`** This is a soft limit and can overflow to other database shared memory heaps such as **`dbheap`**, **`util_heap_sz`**, and **`pckcachesz`**. This cache can also have spill-ins from **`pckcachesz`**.

This parameter is configurable online.

Performance considerations

If this catalog cache is too small, then performance of binding packages, compiling statements, and authorization checking can degrade due to disk I/Os. In addition, catalog cache overflows to other database shared memory heaps can result in overall system performance degradation.

If the catalog cache is too big, then memory would be wasted holding copies of information that is no longer used; this memory would be better served used by other database shared memory heaps, such as buffer pools or the package cache.

Best practices

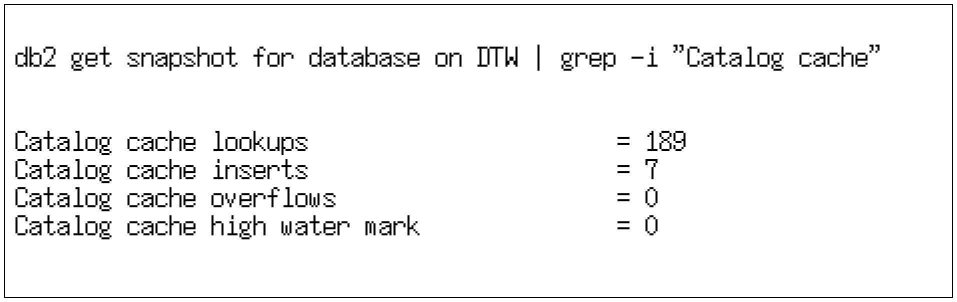
We recommend the following best practices for tuning **`catalogcache_sz`**. For both OLTP and BI environments:

- Start with the default value, and tune its value using the Database System Monitor.
- Strive for a high catalog cache hit ratio of 0.8 or more, and eliminate overflows to other heaps.
- Choose **`catalogcache_sz`** equal to (`Catalog cache high water mark / 4096`) from Figure 3-38.

Performance monitoring metrics

Metrics to monitor lock-related information may be obtained from the snapshot monitor as well as catalog cache indicators in the Health Center, as shown in Figure 3-26 on page 222.

Figure 3-38 on page 274 lists selected fields from the **`get snapshot for db`** command.



```
db2 get snapshot for database on DTW | grep -i "Catalog cache"


Catalog cache lookups                         = 189
Catalog cache inserts                         = 7
Catalog cache overflows                       = 0
Catalog cache high water mark                 = 0
```

Figure 3-38 Database snapshot - catalog cache monitor elements

Important: All the fields in the snapshot monitor, whether they are water marks, counters or gauges, should be monitored over many representative intervals spread over a few weeks, to detect consistent trends before reacting with configuration changes.

Note that counter-type monitoring elements should be reset at the beginning of each monitoring interval by issuing the **`RESET MONITOR`** command.

All fields in Figure 3-38 are of interest in tuning **`catalogcache_sz:`**

- `Catalog cache lookups` is a counter that records the number of times that the catalog cache was referenced to obtain table descriptor information or authorization information. This is used to compute the catalog cache hit ratio.
- `Catalog cache inserts` is a counter that records the number of times that the system tried to insert table descriptor or authorization information into the catalog cache. This is used to compute the catalog cache hit ratio.
- `Catalog cache overflows` is a counter that records the total number of times that the catalog cache overflowed the bounds of its allocated memory.
- `Catalog cache high water mark` is a water mark that records the largest size (in bytes) reached by the catalog cache. If the catalog cache overflowed, then this element contains the largest size reached by the catalog cache during overflow.

Compute the catalog cache hit ratio (CCHR) as follows:

```
CCHR = (1 - (Catalog cache inserts / Catalog cache lookups))
```

Consider *increasing* the value of **`catalogcache_sz`** under the following circumstances:

- The catalog cache hit ratio (CCHR) is below 0.8; use a trial and error method of increasing the size of the catalog cache until CCHR stops increasing.
- `Catalog cache overflows` is greater than zero.

Consider *decreasing* the value of **`catalogcache_sz`** when ((`Catalog cache high water mark`) / `4096`) is consistently less than the value of **`catalogcache_sz`.**

3.4.9 Sort considerations

DB2 may perform sorts to return query results in a desired order (**`SELECT`** statement that uses the **`ORDER BY`** clause) when performing joins and during index creation. The performance of sort depends upon many factors including the writing of efficient SQL, and the configuring of the following parameters:

- Database manager configuration parameter
  - **`sheapthres`** is an instance-wide limit on the total amount of memory that can be consumed for sorts. It is used differently for private and shared sorts[11] as follows:
    - 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 this limit is reached, the memory allocated for additional incoming private sort requests will be considerably reduced. These sorts are called “post-threshold” sorts.
    - For shared sorts, this parameter is a *database-wide* hard limit on the total amount of memory that can be consumed by shared sorts at any given time. When this limit is reached, no further shared-sort memory requests are allowed (they will fail with SQL0955C) until the total shared sort memory consumption falls below the limit specified by **`sheapthres`**.

      Attention: This limit only applies to shared sorts when the **`sheapthres_shr`** database configuration parameter is set to zero.

    The default value is 20000 4 K pages for UNIX and 64-bit platforms, and 10000 4 K pages for Windows.
- Database configuration parameters
  - **`sortheap`** defines the maximum number of private memory pages to be used for a private sort, or the maximum number of shared memory pages to be used for a shared sort. Each sort operation has a separate sort heap

[11] DB2 performs shared sorts in database shared memory when the **`INTRA_PARALLEL`** database manager configuration parameter is enabled; otherwise, it performs private sorts in private agent memory.

that is allocated as needed by DB2 and freed when the sorting completes. In the case of a piped sort (see the definition in "Return of results of from the sort phase" on page 277), the sort heap is not freed until the application closes the cursor associated with the sort.

If directed by the DB2 optimizer, a smaller sort heap than the one specified by **`sortheap`** is allocated by DB2.

The default is 256 4 K pages.

This parameter is configurable online.

– **`sheapthres_shr`** is a *database-wide* hard limit on the total amount of database shared memory that can be used for shared sorts. When this limit is reached, no further shared-sort memory requests are allowed (they will fail with SQL0955C) until the total shared sort memory consumption falls below the limit specified by **`sheapthres_shr`**.

  `sheapthres_shr` is only meaningful in two cases as follows:

  i. 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.

  ii. If the connection concentrator (database manager configuration parameters **`max_connections`** greater than **`max_coordagents`**) is enabled because sorts that use a cursor declared with the WITH HOLD option will be allocated from shared memory.

DB2 sorts are performed in two phases, as follows:

1. Sort phase

   When a sort is performed, DB2 allocates a block of memory equivalent to **`sortheap`** in which data is sorted. When the sort cannot be sorted entirely within the sort heap, it overflows into the buffer pool and temporary table as shown in Figure 3-39 on page 277 and is called an *overflowed sort*. When no overflow occurs, the entire sort is completed in the sort heap and is called a *non-overflowed sort*.

   Attention: Sorts that do not overflow perform better than those that do.

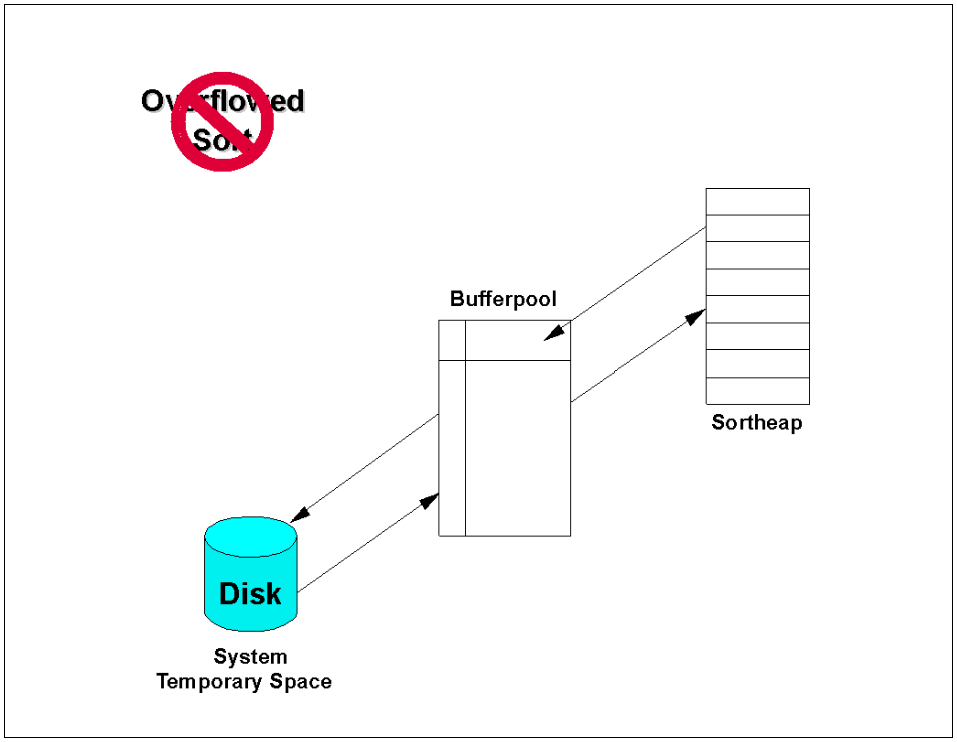


Figure 3-39 Overflowed sorts

2. Return of results of from the sort phase

   If sorted information can return directly without requiring a temporary table to store the final sorted list of data, then it is called a *piped sort*. If the sorted information requires a temporary table to be returned, then it is called a *non-piped sort*. Figure 3-40 on page 278 shows an example of a non-overflowed piped sort.

Attention: A piped sort always performs better than a non-piped sort.

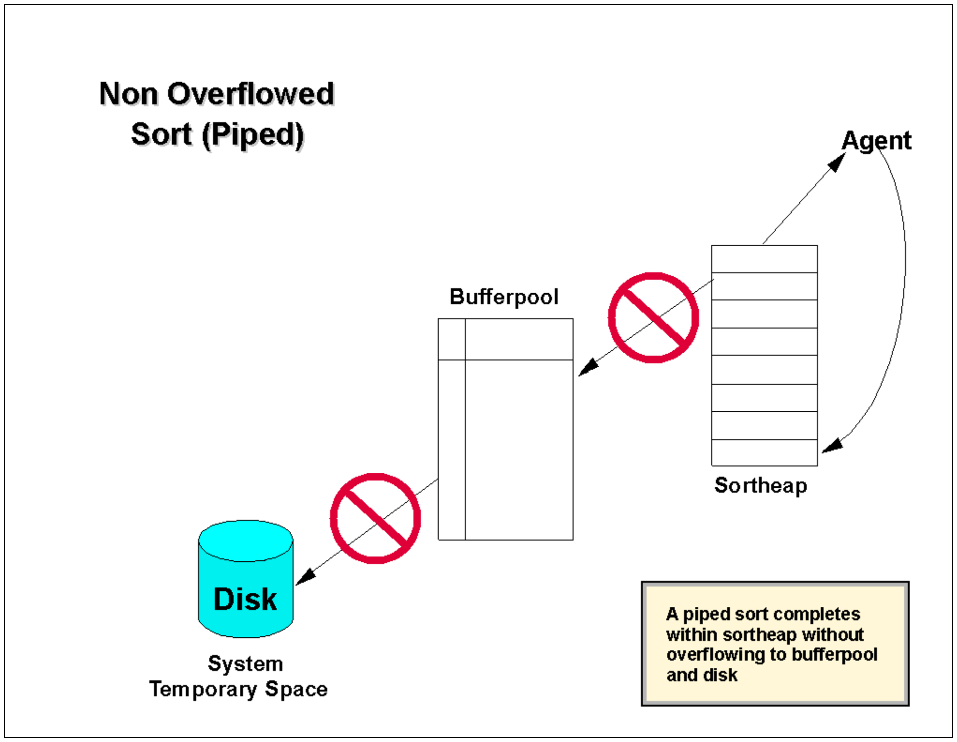

Non Overflowed
Sort (Piped)
Agent
Bufferpool
Sortheap
Disk
System
Temporary Space
A piped sort completes within sortheap without overflowing to bufferpool and disk

Figure 3-40 Non-overflowed piped sorts

Note: The DB2 optimizer will attempt to calculate the size of the sort heap that will be needed based on table statistics. If it requires more space than configured by `sortheap`, then the sort will be overflowed. Otherwise, DB2 will attempt to allocate the entire `sortheap` for the sort. In addition, the DB2 optimizer will also determine whether a piped or non-piped sort should be performed.

Performance considerations

Avoiding sorts is generally preferred through appropriate indexing of tables and writing of efficient SQL (Index SARGable predicates).

If sorts cannot be avoided, then:

- Non-overflowed piped sorts are preferred by configuring an appropriate `sortheap` value.

- Post-threshold sorts should also be avoided by configuring an appropriate **`sheapthres`** value, since they result in smaller sort heap allocations and therefore the potential for overflowed sorts.
- Avoid a rejection of new shared sorts (SQL0955C messages) due to an insufficient configuration of **`sheapthres_shr.`**

Note: Overconfiguring these parameters without adequate real memory to back it up can result in system paging that is detrimental to overall performance, even though non-overflow piped sorts are indicated.

Best practices

We recommend the following best practices for enhancing sort performance and tuning the **`sortheap`**, **`sheapthres`** and **`sheapthres_shr`** parameters:

1. Avoid sorts as far as possible by defining appropriate indexes on tables and writing efficient SQL. Use the Design Advisor wizard or **`db2advis`** command and EXPLAIN described in 2.6.7, "Design Advisor" on page 93 and EXPLAIN as described in 2.6.8, "Explain and Visual Explain" on page 94 against all long running queries to verify index access.
2. Start with the default values, and tune **`sortheap, sheapthres,`** and **`sheapthres_shr`** to minimize overflows and non-piped sorts and then tune them for optimal values.
   – For critical workloads where frequent large sorts are performed, consider setting up a representative workload and tuning **`sortheap, sheapthres,`** and **`sheapthres_shr`** for that workload before adopting it in the production environment.
3. For OLTP environments:
   – Disable the **`INTRA_PARALLEL`** database manager configuration parameter.
   – Ensure that very few if any sorts are performed.
4. For BI environments:
   – Enable the **`INTRA_PARALLEL`** database manager configuration parameter.
   – Increase **`sheapthres`** parameter and **`sheapthres_shr`** because of the increased number and size of sorts.

Performance monitoring metrics

Metrics for monitoring **`sortheap, sheapthres`** and **`sheapthres_shr`** may be obtained from the snapshot monitor and appropriate sorting health indicators in the Health Center as shown in Figure 3-26 on page 222.

Figure 3-41 shows sort relevant snapshot information from the **`get snapshot for dbm`** command, and Figure 3-42 shows sort relevant information from the **`get snapshot for db`** command.

```
db2 get dbm snapshot

Private Sort heap allocated                       = 0
Private Sort heap high water mark                 = 4536
Post threshold sorts                              = 0
Piped sorts requested                             = 143
Piped sorts accepted                              = 143
```

Figure 3-41 Database manager snapshot - sort monitor elements

```
db2 get snapshot for database on DTW


              Database Snapshot

Database name                                 = DTW

Total Private Sort heap allocated             = 0
Total Shared Sort heap allocated              = 0
Shared Sort heap high water mark              = 0
Total sorts                                   = 18
Total sort time (ms)                          = 18086
Sort overflows                                = 8
Active sorts                                  = 0

Commit statements attempted                   = 68
Rollback statements attempted                 = 26
```

Figure 3-42 Database snapshot - sort monitor elements

Important: All fields in the snapshot monitor, whether they are water marks, counters or gauges, should be monitored over many representative intervals spread over a few weeks, to detect consistent trends before reacting with configuration changes.

Note that counter-type monitoring elements should be reset at the beginning of each monitoring interval by issuing the **`RESET MONITOR`** command.

The following fields are of interest for tuning **`sortheap`**, **`sheapthres`**, and **`sheapthres_shr`**:

- Database manager snapshot
  - **`Private sort heap high water mark`** is a water mark of the total number of allocated pages of sort heap space for all private sorts across all databases.
  - `Post threshold sorts` is a counter that records the total number of sorts that have requested heaps after the sort heap threshold (**`sheapthres`**) has been exceeded.
  - `Piped sorts requested` is a counter that records the total number of piped sorts that have been requested.
  - `Piped sorts accepted` is a counter that records the total number of piped sorts that have been accepted. A piped sort is rejected if the sort heap threshold (**`sheapthres`**) will be exceeded when the sort heap is allocated for the sort.
- Database snapshot
  - `Total Private Sort heap allocated` is a gauge that records the total number of allocated pages of sort heap space for all private sorts.
  - `Total Shared Sort heap allocated` is a gauge that records the total number of allocated pages of sort heap space for all shared sorts.
  - `Shared Sort heap high water mark` is a water mark of the total number of allocated pages of sort heap space for all shared sorts for this database.
  - `Total sorts` is a counter that records the total number of sorts that have been executed.
  - `Total sort time (ms)` is a counter that records the total elapsed time in milliseconds for all sorts that have been executed.
  - `Sort overflows` is a counter that records the total number of sorts that ran out of sort heap and may have required disk space for temporary storage.
  - `Active sorts` is a gauge that records the number of sorts in the database that currently have a sort heap allocated.
  - `Commit statements attempted` is a counter that records the total number of commits that have been attempted.
  - `Rollback statements attempted` is a counter that records the total number of rollbacks that have been attempted.

Note: Database manager and database snapshots only indicate the presence of a sort problem, not the SQL statements causing them. In order to identify the SQL statements invoking sorts, a dynamic SQL snapshot or an Event Monitor for SQL statements is required.

Consider modifying **`sortheap`**, **`sheapthres`**, and **`sheapthres_shr`** under the following circumstances:

1. Compare the water marks for Private Sort heap high water mark and Total Shared Sort heap high water mark with the corresponding values of **`sheapthres`** and **`sheapthres_shr`**.

   If they are significantly lower than the configuration parameters, consider setting the configuration parameters to a few percentage points above the water mark levels.

   If Private Sort heap high water mark is higher than **`sheapthres`**, then it means that less than desired amount of sort heap is being allocated; this should register as Post threshold sorts. Increase the **`sheapthres`** value a few percentage points above the water mark.

   If Shared Sort heap high water mark is close or equal to the **`sheapthres_shr`** configuration parameter setting, increase its value a few percentage points above the water mark. User applications may receive the SQL0955C error message.
2. If the percentage of overflowed sorts (Sort overflows / Total sorts) is high, increase **`sortheap`** and/or **`sheapthres`**.
3. If Post threshold sorts relative to Total sorts are high, increase **`sheapthres`** and/or decrease **`sortheap.`**

   Decreasing **`sortheap`** may help since smaller allocations are less likely to cause the **`sheapthres`** threshold to be exceeded. However, decreasing **`sortheap`** may cause sort overflows to occur which is undesirable.
4. If piped sorts rejected (Piped sorts requested - Piped sorts accepted) relative to Piped sorts requested is high, increase **`sortheap`** or **`sheapthres`**.
5. If there are user complaints about receiving SQL0955C error messages which indicates that the shared memory sorts threshold has been exceeded, increase **`sheapthres_shr`**.
6. OLTP environments should have very few sorts occurring, if any, and small ones at that:
   - Compute the sorts per transaction = (Total sorts/(Commit statements attempted + Rollback statements attempted))

If this ratio is greater than 5, it might indicate that there are too many sorts occurring.

  - If the percentage of overflowed sorts (Sort overflows / Total sorts) is greater than 3%, then it indicates that large sorts may be occurring.
  - If the average sort time (Total sort time (ms) / Total sorts) is in seconds, it highlights a potential problem that requires adding appropriate indexes to eliminate sorts, or modifying the problem SQL statements (through dynamic SQL snapshots or Event Monitors).

7. BI environments will traditionally have a number of small and large sorts, and the settings of **`sortheap`**, **`sheapthres`**, and **`sheapthres_shr`** parameters as well as efficient SQL coding, appropriate buffer pool assignments, and proper temporary table space placement will have a significant impact on sort performance.

3.4.10 Other memory considerations

There are a number of other memory heaps not covered in the previous sections that also have an impact on system performance, and should be tuned for optimal performance.

This section discusses some of the more important memory heaps by the category defined in Figure 2-7 on page 32 as follows:

1. Database shared memory
   - Database heap — **`dbheap`**
2. Agent private memory
   - Application heap size —- **`applheapsz`**
   - Maximum java interpreter heap size — **`java_heap_sz`**
   - Query heap size — **`query_heap_sz`**
   - Statement heap size — **`stmtheap`**
3. Agent/Application shared memory
   - Application support layer heap size — **`aslheapsz`**
   - Client I/O block heap size — **`rqrioblk`**

Attention: The first rule of thumb for setting memory allocation parameters is never to set them at their highest values unless such a value can be carefully justified. Many parameters that affect memory can allow database manager easily and quickly to take up all the available memory on a machine.

dbheap

This database configuration parameter allocates space in database shared memory and holds space for many items including:

- Temporary memory for utilities.
- Event Monitor buffers.
- Log buffers as specified by the database configuration parameter **`logbufsz.`**
- Temporary overflows (spill-in) from package cache (database configuration parameter **`pckcachesz`**) and from catalog cache (database configuration parameter **`catalogcache_sz`**).

The minimum amount the database manager needs to get started is allocated at first connection, and then expanded as needed up to the maximum specified by **`dbheap`**.

The default is **`AUTOMATIC`**, and this parameter is configurable online.

Performance considerations

If there is insufficient database heap specified, applications and utilities with fail with messages such as SQL0956C, SQL1084C and SQL2009C. Overconfiguring **`dbheap`** may cause problems allocating the application control shared memory with messages such as SQL0987C.

Best practices

We recommend the following best practices for tuning **`dbheap`**:

1. For critical OLTP and BI environments, determine the appropriate value through benchmarking representative workloads.
2. For most environments, choose the default and let DB2 determine the size and then tune for optimal value.
3. Ensure that **`dbheap`** is increased correspondingly whenever any of its component heaps are increased, such as **`logbufsz.`**

Performance monitoring metrics

Metrics to monitor the **`dbheap`** may be obtained from the snapshot monitor as well as the Database Heap Utilization health indicator in the Health Center as shown in Figure 3-26 on page 222.

Figure 3-43 shows selected fields of the **get snapshot for db** command.

```
Memory Pool Type                              = Database Heap
   Current size (bytes)                       = 1638400
   High water mark (bytes)                    = 1671168
   Maximum size allowed (bytes)               = 9846784
```

Figure 3-43 Database snapshot - database heap memory usage

Important: All the fields in the snapshot monitor, whether they are water marks, counters or gauges, should be monitored over many representative intervals spread over a few weeks, to detect consistent trends before reacting with configuration changes.

Note that counter-type monitoring elements should be reset at the beginning of each monitoring interval by issuing the **RESET MONITOR** command.

The fields of interest in Figure 3-43 for tuning **dbheap** are:

- `High water mark (bytes)` is a water mark that of the highest utilization of memory by the database heap.
- `Maximum size allowed (bytes)` is the maximum amount of memory as specified by the **dbheap** parameter.

Consider doubling the value of **dbheap** when:

- `High water mark (bytes)` is around 90% utilization of the `Maximum size allowed (bytes)` value.
- When applications and utilities report memory shortages via SQL0956C, SQL1084C, and SQL2009C messages.

In the case of the SQL2009C message using the DB2 backup utility, first check the database configuration parameter **util_heap_sz** and the backup utility's **buffers** value specified in the backup command and tune them accordingly before increasing the **dbheap** value.

applheapsz

This database configuration parameter allocates space in agent private memory and defines the number of pages available to be used by the database manager on behalf of a specific agent or subagent. This heap is used to store copies of the executing sections of SQL statements for agents and subagents.

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. The default value is 256 4 K pages.

This parameter is *not* an online configurable parameter, hence the database needs to be deactivated and reactivated for changes to take effect.

Performance considerations

Under configuring this parameter will result in the application receiving SQL0945C error messages complaining about inadequate application heap size. Overconfiguring **`applheapsz`** causes overallocation of agent private memory, even though **`applheapsz`** is allocated on demand.

Best practices

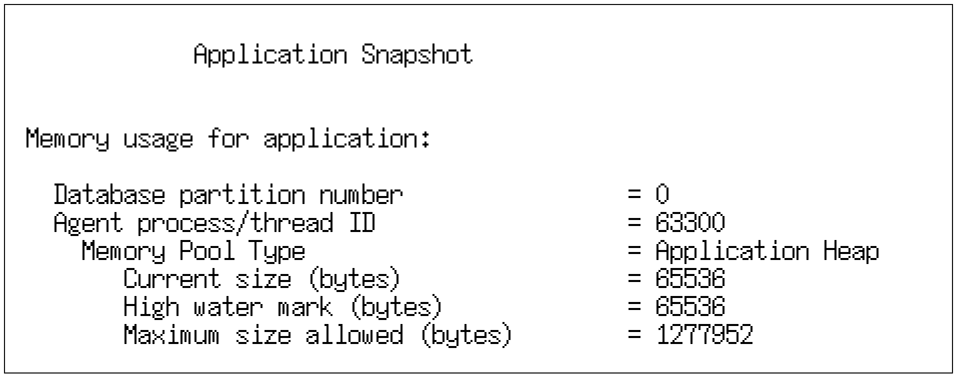
We recommend the following best practices for tuning **`applheapsz:`**

1. For critical OLTP environments, determine the appropriate value through benchmarking representative workloads.
2. For most environments, choose the default and then tune to an optimal value.

Performance monitoring metrics

Metrics to monitor the **`applheapsz`** may be obtained from the snapshot monitor as well as the Memory Visualizer as described in 2.6.11, “Memory Visualizer” on page 103.

Figure 3-44 lists selected fields from the **`get snapshot for applications`** command.



```
            Application Snapshot


Memory usage for application:

  Database partition number               = 0
  Agent process/thread ID                 = 63300
    Memory Pool Type                      = Application Heap
       Current size (bytes)               = 65536
       High water mark (bytes)            = 65536
       Maximum size allowed (bytes)       = 1277952
```

Figure 3-44 Application snapshot - application memory usage by all agents

Important: All the fields in the snapshot monitor, whether they are water marks, counters or gauges, should be monitored over many representative intervals spread over a few weeks to detect consistent trends before reacting with configuration changes.

Note that counter-type monitoring elements should be reset at the beginning of each monitoring interval by issuing the **`RESET MONITOR`** command.

The fields of interest in Figure 3-44 on page 286 for tuning **`applheapsz`** are:

- `High water mark (bytes)` is a water mark that of the highest utilization of memory by the application heap.
- `Maximum size allowed (bytes)` is the maximum amount of memory as specified by the **`applheapsz`** parameter.

Consider increasing the value of **`applheapsz`** when:

- `High water mark (bytes)` is around 80% to 90% utilization of the `Maximum size allowed (bytes)` value.
- When applications report memory shortages via SQL0945C messages.

Consider decreasing the value of **`applheapsz`** when the water mark is consistently below the maximum specified.

java_heap_sz

This database manager configuration parameter determines the size of the heap that will be used by the Java interpreter started to service Java DB2 stored procedures and UDFs.

This heap is allocated in full when a Java stored procedure or UDF starts, and freed when the db2fmp process (fenced) or the db2agent process (trusted) terminates.

The heap is allocated as follows.

- One heap for each fenced UDFs and fenced stored procedure process
- One heap per agent not including sub-agents for trusted routines
- One heap per db2fmp process running Java stored procedure
- Multi-threaded db2fmp process service multiple applications from one heap

In all situations, only the agents or processes that run Java UDFs or stored procedures ever allocate this memory.

The default is 512 4 K pages, and this parameter is *not* configurable online.

Performance considerations

Underconfiguring this parameter will result in applications failing with an error message such as SQL4301N. Overconfiguring **`java_heap_sz`** causes overallocation of agent private memory.

Best practices

We recommend the following best practices for tuning **`java_heap_sz`**:

1. For critical OLTP and BI environments, determine the appropriate value through benchmarking representative workloads.
2. For most environments, choose the default and tune for optimal value using a trial and error method.

Performance monitoring metrics

There are no metrics for monitoring this parameter.

When applications receive SQL4301N messages, the **`java_heap_sz`** should be increased incrementally until the applications no longer receive the SQL4301N message.

query_heap_sz

This database manager configurable 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 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`** database configuration parameter. When an application connects to DB2, the initial size of the query heap is the minimum of two pages, or equal to the size specified by the **`aslheapsz`** database configuration 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 as long as it does not exceed **`query_heap_sz`**. If this revised query heap value is more than 1.5 times the size of **`aslheapsz`**, then the query heap will be reallocated to the size of **`aslheapsz`** when the query ends.

The query heap is allocated when an application connects to the database, and is deallocated when the application disconnects from the database or detaches from the instance.

The default value is 1000 4 K pages, and this parameter is *not* configurable online.

Performance considerations

Underconfiguring this parameter will cause applications to fail with messages such as SQL0973N. Overconfiguring **query_heap_sz** causes overallocation of agent private memory even though **query_heap_sz** is allocated on demand.

Best practices

We recommend the following best practices for tuning **query_heap_sz**:

1. For critical OLTP and BI environments, determine the appropriate value through benchmarking representative workloads.
2. For most environments, choose a value that is at least five times as large as the **aslheapsz** parameter value to allow for queries that are larger than **aslheapsz** and to allow for enough memory to support three to four concurrent blocking cursors.

Note: If applications access LOBs without using LOB locators, the query heap may need to be increased using a trial and error method.

Performance monitoring metrics

There are no metrics for monitoring this parameter; however, see monitoring metrics for “aslheapsz” on page 291.

When applications receive SQL0937N messages, the **query_heap_sz** should be increased incrementally until the applications no longer receive the SQL0937N message.

stmtheap

This database configuration parameter specifies the size of the workspace to be used by the SQL compiler during SQL compilation. Memory corresponding to **stmtheap** is allocated and released for every SQL statement handled.

- For dynamic SQL statements, this area is used during execution of the program.
- For static SQL statements, this area is used only during bind process.

The default value is 2048 4 K pages, and this parameter is configurable online.

Performance considerations

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.

Setting **`stmtheap`** to a low value could result in sub-optimal plans if the DB2 optimizer has insufficient space to optimize the statement. In this case, the optimizer may automatically drop to use a lower optimization class set of strategies; a +437 (SQL0437W) warning with a reason code of 1 is given in such cases. If it is not possible to compile or bind the statement with the current **`stmtheap`** size, then the SQL0101N error is issued. Overconfiguring **`stmtheap`** can result in wasted memory that could be better used in other heaps.

Best practices

We recommend the following best practices for tuning **`stmtheap`**:

1. For OLTP applications, where the SQL is fairly simple, the default value of 2048 4 KB pages should be sufficient.
2. For BI environments with very complex SQL statements, consider 4096 to 8192 pages as a good starting point and then tune it for optimal value.

Performance monitoring metrics

There are no metrics for monitoring this parameter.

When precompilations and binds receive SQL0101N messages, the **`stmtheap`** should be increased in increments of 1024 pages until the SQL0101N messages no longer appear.

Sometimes the query may be too complex to be compiled successfully with even very large statement heaps; in this case, the query will need to be broken up into smaller parts.

`db2_reduced_optimization` is a registry variable that tells the optimizer to avoid certain optimization strategies at optimization class 5; the default setting is NO. 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. Although optimization time and resource use might be reduced, the risk of producing a less than optimal data access plan is increased. The strategies that are not considered are typically those that consume much time and may not have a significant impact on the plan in most cases.

Consider enabling this parameter if compile time and memory used is a major issue.

aslheapsz

This database manager configuration parameter specifies the application support layer heap size, which represents a communication buffer between a *local* application and its associated agent. This buffer is allocated as agent/application shared memory by each database manager agent is started.

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.

`aslheapsz` is allocated when the database manager agent process is started for the local application, and is deallocated when the database manager agent process is terminated.

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 default is 15 4 K pages, and this parameter is not configurable online.

Performance considerations

If the request to the database manager, or its associated reply, does 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.

You should also consider the effect of this parameter on the number and potential size of blocking cursors. Large row blocks may yield better performance if the number or size of rows being transferred is large (for example, if the amount of data is greater than 4096 bytes). However, the trade-off is that the 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 as described in “OPTIMIZE FOR n ROWS clause” on page 181.

Note: 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, and other fields which do not generally exceed 250 bytes.

Best practices

We recommend the following best practices for tuning **aslheapsz**:

1. 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.
2. Compute **aslheapsz** as follows if the information is available:

   ```
   aslheapsz = ((size of input SQLDA) + (size of each input SQLVAR) + (size
   of output SQLDA) + 250))/4096
   ```

   where “size of (x)” is the size of x in bytes that calculates the number of pages of a given input or output value.
3. Allow **aslheapsz** to default to 15 4K pages, and then tune for optimal value.

Performance monitoring metrics

DB2 does not provide a direct mechanism to monitor and tune **aslheapsz**. You instead need to monitor other elements in an application snapshot which indirectly relate to **aslheapsz**. The monitoring elements of interest in an application snapshot as generated by a **get snapshot for applications** command are shown in Example 3-14.

Example 3-14 Snapshot showing block remote cursor information

```
Rejected Block Remote Cursor requests      = 50
Accepted Block Remote Cursor requests      = 1000
```

Important: All fields in the snapshot monitor, whether they are water marks, counters or gauges, should be monitored over many representative intervals spread over a few weeks, to detect consistent trends before reacting with configuration changes.

Note that counter-type monitoring elements should be reset at the beginning of each monitoring interval by issuing the **RESET MONITOR** command.

The fields of interest in Example 3-14 on page 292 for tuning **aslheapsz** is:

- `Rejected Block Remote Cursor requests` is a counter that records the number of times a request for an I/O block at the database server was rejected and the request was converted to non-blocked I/O.

  If there are many cursors blocking data, the communication heap may become full. When this heap is full, an error is *not* returned; instead, no more I/O blocks are allocated for blocking cursors. If cursors are unable to block data, performance can be affected.

- `Accepted Block Remote Cursor requests` is a counter that records the number of times a request for an I/O block at the database server was accepted.

Compute the following metric for tuning purposes:

Percentage of rejected block remote requests (PRBRR) as follows:

`PRBRR = ((Rejected Block Remote Cursor requests) / (Accepted Block Remote Cursor requests + Rejected Block Remote Cursor requests)) * 100`

Consider increasing **aslheapsz:**

1. If **PRBRR** is consistently high, then **query_heap_sz** should also be increased correspondingly.
2. When applications receive SQL1221N or SQL1222N messages, or see DIA3605C in the db2diag.log, *then* increase **aslheapsz** incrementally until these conditions do not reappear.

The number of send and receive requests occurring for a particular application can be found by enabling the CLI trace facility. Refer to 2.6.1, “CLI/ODBC/JDBC trace” on page 67 for more information on CLI trace.

rqrioblk

This database manager configuration parameter specifies the size of the communication buffer between *remote* applications and their database agents on the database server. When a database client requests a connection to a remote database, this communication buffer is allocated on the client. On the database server, a communication buffer of 32767 bytes is initially allocated, until a connection is established and the server can determine the value of **rqrioblk** at the client.

Once the database server knows this value, it will reallocate its communication buffer if the client’s buffer is not 32767 bytes. In addition to this communication buffer, this parameter is also used to determine the I/O block size at the database client when a blocking cursor is opened. This memory for blocked cursors is allocated out of the application's private address space, so you should determine the optimal amount of private memory to allocate for each application program. If the database client cannot allocate space for a blocking cursor out of an application's private memory, a non-blocking cursor will be opened.

The communication buffer is allocated when:

- A remote client application issues a connection request for a server database.
- A blocking cursor is opened, additional blocks are opened at the client.

The communication buffer is deallocated when:

- The remote client application disconnects from the server database.
- The blocking cursor is closed.

The default value is 32767 bytes, and this parameter is *not* configurable online.

Performance considerations

The same considerations discussed with **`aslheapsz`** in "Performance considerations" on page 291 apply here as well.

> **Note:** **`aslheapsz`** relates to local application connections, while **`rqrioblk`** applies to remote connections.

Best practices

The same considerations discussed with **`aslheapsz`** in "Best practices" on page 292 apply here as well.

Performance monitoring metrics

The same considerations discussed with **`aslheapsz`** in "Performance monitoring metrics" on page 292 apply here as well.

3.4.11 Miscellaneous considerations

There are a number of other, non-memory related considerations not explicitly covered in the previous sections that also have an impact on system performance, and should be tuned for optimal performance.

This section discusses some of the more important considerations as follows:

- **`reorg`** objects
- **`runstats`** collection
- **`intra_parallel`** and **`max_querydegree`** database manager configuration parameters
- **`agentpri`** database manager configuration parameter

- **maxfilop** database configuration parameter
- Configuration Advisor

reorg objects

The **reorg** utility re-establishes clustering and free space, and recovers unused space. This provides good performance improvement and should be run whenever a database object has become disorganized.

To determine whether a **reorg** is appropriate, DB2 provides a **reorgchk** utility that calculates statistics on the database using eight different formulas to determine if tables or indexes or both require **reorg** to be run.

Figure 3-45 shows the results of the **reorgchk** utility after it has invoked **runstats** to update table statistics. The -** under the REORG column indicates that the computed values for F2 and F3 for table **DB2INST1.CONN_OLTP_CONNECTION** (the > symbol in the name represents the truncated portion of the name of the table) exceed the bounds set for these formulas, indicating the need for a reorganization. Similarly, the --* for the table **DB2INST1.ADVISE_INDEX** indicates that the computed value for F3 exceeds the set bounds.

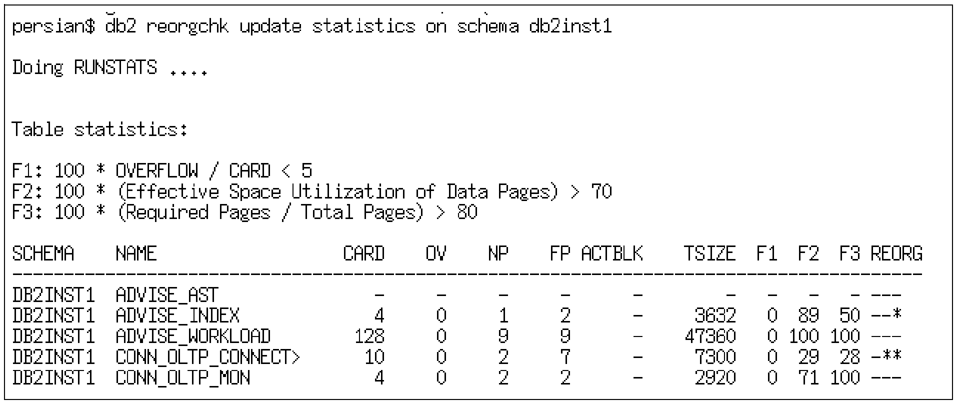

```
persian$ db2 reorgchk update statistics on schema db2inst1

Doing RUNSTATS ....


Table statistics:

F1: 100 * OVERFLOW / CARD < 5
F2: 100 * (Effective Space Utilization of Data Pages) > 70
F3: 100 * (Required Pages / Total Pages) > 80

SCHEMA    NAME                  CARD    OV    NP    FP ACTBLK    TSIZE  F1  F2  F3 REORG
----------------------------------------------------------------------------------------
DB2INST1  ADVISE_AST               -     -     -     -      -        -   -   -   - ---
DB2INST1  ADVISE_INDEX             4     0     1     2      -     3632   0  89  50 --*
DB2INST1  ADVISE_WORKLOAD        128     0     9     9      -    47360   0 100 100 ---
DB2INST1  CONN_OLTP_CONNECT>      10     0     2     7      -     7300   0  29  28 -**
DB2INST1  CONN_OLTP_MON            4     0     2     2      -     2920   0  71 100 ---
```

Figure 3-45 Output of REORGCHK command

For more information on **reorgchk**, and the formulas (F1 through F8), refer to *DB2 UDB Command Reference*, SC09-4828, and to 4.6, “Reorg” on page 321 for information about tuning the performance of the **reorg** utility.

runstats collection

Critical to the DB2 optimizer choosing an optimal access path for a query is the need for up-to-date metadata about the target objects in the catalog tables.

Generally, **runstats** should be run whenever significant changes have occurred to the objects; this includes any of the following events:

- After the data has been initially loaded and statistics have never been run on the table and indexes.
- When a new index is created on a populated table.

  Note: Statistics may be collected during index creation using the **COLLECT ....STATISTICS** option.

- When the table and index characteristics are changed using an **ALTER** statement such as changing the prefetch size.
- After “significant” updates (INSERT/UPDATE/DELETE) have occurred to the table. The percentage change may be any value upwards of 10%; a trial and error mechanism may pinpoint the optimal percentage for a given object based on the access path impact it has on queries accessing that object.
- After **reorg** is run on the table or index.
- After **load** is run on the table.

The options chosen must depend on the specific table and the application, and may reduce the time it takes to collect statistics. In general:

- If the table is a very critical table in critical queries, is relatively small, or does not change too much and there is not too much activity on the system itself, then it may be worth spending the effort to collect statistics in as much detail as possible.
- If the time to collect statistics is limited, and:
  - The table is relatively large and/or changes a lot, it might be beneficial to execute **runstats** limited to the set of columns that are used in predicates. This way, you will be able to execute the **runstats** command more often.
  - The effort to tailor the **runstats** command on a table by table basis is a major issue, consider collecting statistics for the KEY columns only. It is assumed that the index contains the set of columns that are critical to the table and are most likely to appear in predicates.
- If there are many indexes on the table and collecting **DETAILED** (extended) information on the indexes seems like a good idea, the **SAMPLED** option might be considered in order to cut down on the time it takes to collect statistics—this is very important in BI environments.
- If there is skew in certain columns and predicates of the type column = constant, it may be beneficial to specify a larger **NUM_FREQVALUES** value for that column

- For columns that have range predicates (for example column >= constant, column BETWEEN constant1 AND constant2), it may be beneficial to specify a larger **`NUM_QUANTILES`** value.
- If storage space is a concern and you cannot afford too much time on collecting statistics, do not specify high **`NUM_FREQVALUES`** or **`NUM_QUANTILES`** values for columns that are not used in predicates.

If index statistics are requested, and statistics have never been run on the table containing the index, statistics on both the table and indexes are calculated.

Attention: Unlike the case of **`reorg`** with the **`reorgchk`** utility, determining whether "significant" changes have occurred to a specific object in order to trigger **`runstats`** is a non-trivial exercise, and requires routine monitoring activity over time to identify the occurrence of events that need to trigger **`runstats`**.

For more information on **`runstats`**, refer to *DB2 UDB Command Reference*, SC09-4828, and to 4.8, "Runstats" on page 326, for information about tuning the performance of the **`runstats`** utility.

Intra_parallel and max_querydegree dbm cfg parameters

The **`intra_parallel`** database manager configuration parameter specifies whether the database manager can use intra-partition parallelism[12] for a query, index creation, and utilities such as database load.

Attention: **`intra_parallel`** should only be enabled when the database server has multiple CPUs.

Besides enabling **`intra_parallel,`** set the **`dft_degree`** database configuration parameter to **`-1`** or **`ANY`**, which 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 of parallelism 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 in order to minimize adverse performance impact on other database users.

[12] A query is divided into parts and executed in parallel at the same time within the database partition if this parameter is enabled.

Another database configuration parameter that has an impact on query performance is **max_querydegree,** which specifies the maximum degree of intra-partition parallelism that may be used for any SQL statement executing at database manager level. This parameter should be set to the number of CPUs in the system to avoid the possibility of users inadvertently or intentionally setting their **CURRENT DEGREE** register value or **DEGREE** bind option too high.

Best practices

We recommend the following best practices for tuning **intra_parallel**, **dft_degree**, and **max_querydegree**:

1. For OLTP environments, set **intra_parallel** to **NO** since transactions query small amounts of data, and parallelism would hurt rather than improve performance.
2. For BI environments with the number of active users less than the number of CPUs, set **intra_parallel** to **YES**, **dft_degree** to **-1**, and **max_querydegree** to the number of CPUs, since queries tend to access very large amounts of data and parallelism could significantly enhance performance.

agentpri

This database manager configuration parameter controls the priority given to all agents, and other database manager instance processes and threads, by the operating system scheduler. This includes both coordinating agents and subagents, the parallel system controllers, and the FCM daemons when intra-partition parallelism is enabled and/or the database is partitioned. 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 (the default), 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 this parameter. This allows you to control the priority with which the database manager processes and threads will execute on your machine, and potentially 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.

The default is -1, and this parameter is *not* configurable online.

Performance considerations

For some dedicated database servers and non-dedicated database servers in particular, increasing the priority of DB2 processes and threads may result in significant performance benefits. However, this should be done with care because the performance of other user processes can be severely degraded, especially at very high CPU utilizations. The setting of this parameter should be balanced with the other concurrent activity expected on the database server.

Best practices

We recommend the following best practices for tuning `agentpri`:

1. For dedicated database servers, the default value should provide good performance since it provides a good compromise between response time to other users/applications and database manager throughput.
2. For performance-critical OLTP and BI environments on dedicated or non-dedicated database servers, consider using benchmarking techniques to determine the optimum setting for this parameter. Increasing the priority of the database manager processes and threads can have significant performance benefits. However, it is necessary to benchmark both DB2 and user/applications at different `agentpri` to get a optimal value.

> **Note:** If you set this parameter to a non-default value on UNIX-based platforms, you cannot use the DB2 governor to alter agent priorities.

Performance monitoring metrics

DB2 does not have a facility to monitor CPU utilization of DB2 processes and threads; you need to use operating system tools for this purpose.

On AIX, consider using **vmstat, sar, top** for getting CPU information of DB2 process as described in 5.2.4, “Monitoring and problem determination tools” on page 363, and the task manager in Windows as described in 5.3.4, “Monitoring and problem determination tools” on page 387.

In tuning `agentpri` in a non-dedicated database server machine, investigate sustained conditions where non-DB2 processes dominate CPU consumption vis-a-vis DB2 processes and performance problems are being reported, especially in a CPU resource-constrained[13] environment. If CPU resource

[13] Typically, you should aim for server steady state CPU utilization of less than 70%, as this provides space capacity to cope with unexpected peak workloads.

constraints cannot be alleviated through additional hardware, consider adopting a trial and error approach to increasing **`agentpri`** in incremental steps until DB2 performance improves to your satisfaction. However, this will most likely degrade the performance of non-DB2 processes, and appropriate trade-offs will have to be made.

maxfilop

This database configuration parameter specifies the maximum number of file handles that can be open for each database agent.

Note: Any process is limited to opening a certain number of files. This limit is controlled by the **`ulimit`** command.

`ulimit -a` displays a quantity **nofiles(descriptors)**, which is this limit. This value can be changed.

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.

SQL applications access one or more tables which in turn may use indexes, all of which reside in SMS or DMS table spaces. 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.

The database manager opens files for reading and writing into and out of the buffer pool. If the limit is exceeded, one file will be closed before the new file is opened. DB2 attempts to keep them open throughout to minimize the overhead of opening and closing these files during SQL execution.

The default value is 64, and this parameter is configurable online.

Performance considerations

If opening a file by a database agent 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. Having too many files open that are rarely re-accessed consumes memory that may be better utilized elsewhere.

There is also another database manager configuration parameter **`maxtotfilop`**[14] (does not apply to UNIX systems) which places a limit on the maximum number of files that can be opened by all agents and other threads executing in a single

[14] The default is 16000, with a maximum value of 32768; this parameter is not configurable online.

database manager instance. If opening a file causes this value to be exceeded, an SQL0931C error message is returned to the application. Therefore, having too large a **maxfilop** with many concurrent agents can cause this limit to be exceeded resulting in applications failing with the SQL0931C error message.

Best practices

We recommend the following best practices for tuning **maxfilop:**

1. The default value of 64 is unacceptable for most databases.

   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.

   A value of 1500 is a good starting point, which can then be tuned for an optimal value through trial and error.

   Note: The upper limit for UNIX is 1950; the upper limit for Windows is 32768.

2. The default value of 16000 for **maxtotfilop** is far too small for most databases. Compute an appropriate number based on the maximum number of concurrent applications (**maxappls**) and the estimated number of file handles that could be opened by each application and choose this value to be as follows:

   ```
   maxtotfilop = maxappls * maxfilop
   ```

   Ensure that (**maxappls * maxfilop**) does *not* exceed the operating system limit on open files.

Performance monitoring metrics

DB2 only monitors the total number of files closed and does not provide a high water mark of open files desired. It also does not monitor whether the **maxtotfilop** limit was exceeded.

For non-UNIX systems, since applications receive the SQL0931C error message when **maxtotfilop** is exceeded, this value must be incrementally increased until there is not recurrence of the occurrence of the SQL0931C error message.

For **maxfilop**, monitoring metrics may be obtained from the g**et snapshot for bufferpools** command, as shown in Figure 3-32 on page 255.

The field of interest in tuning **maxfilop** is:

- `Database files closed` is a counter that records the total number of files closed.

Consider increasing `maxfilop` incrementally until this field shows a zero value.

Configuration Advisor

The Configuration Advisor, as described in 2.5.1, “Configuration Advisor and AUTOCONFIGURE” on page 54, provides a facility to set initial values for many of the key database manager and database configuration parameters of existing systems. However, monitoring key performance metrics and tuning them is key to achieving optimal performance.

4

Command and utility considerations

In this chapter we describe key command and utility considerations, and their best practices for achieving superior DB2 performance.

The commands and utilities covered are:

- Backup
- Export
- Import
- Load
- Reorg
- Restore
- Runstats

4.1 Introduction

The objective of this chapter is to concentrate on identifying key considerations in improving the performance of some of the main commands and utilities, and providing best practices guidelines for optimizing the performance of their execution.

Attention: We assume that the DBA is familiar with the functionality of the individual commands and utilities, and the considerations in using a specific available feature or option; therefore, we do not discuss them here.

For example, we will not discuss the pros and cons of choosing between a full, incremental, or delta table space copy, but we do discuss considerations for improving the performance of full, incremental, and delta table space copies after the choice has been made.

For each command/utility of interest, we provide the following:

- A brief description of the command/utility
- A description of factors that impact the performance of the command/utility
- Best practices guidelines for achieving better performance

Important: With utilities in general, the objective is to reduce elapsed times and CPU consumption. For offline utilities, it may be appropriate to focus on reducing elapsed times while maximizing parallelism and CPU consumption. Few monitoring elements are available in DB2 to monitor and tune the performance of utilities, although some utilities such as import and export use SQL that is instrumented with monitoring elements that can be exploited for performance monitoring and tuning of these utilities.

Therefore, DBAs should rely on operating system facilities to monitor DB2 utility execution and employ a trial and error approach to tuning utilities for optimal performance.

4.2 Backup

The `backup` command is used in conjunction with the `restore` command to ensure recoverability of a database and/or table space.

The following subsections are organized as follows:

- Brief description
- Performance considerations
- Best practices

4.2.1 Brief description

The DB2 **`backup`** command supports the creation of full, incremental, and delta backup images of databases and table spaces. It also provides support for the use of parallelization during the backup process. A record of the backups is kept in a recovery history file, and includes information such as the part of the database that was copied, the type of copy, the time the copy was made, the location of the copy, and the last log sequence number saved by the database backup. The content of the recovery history file may be viewed by issuing the **`list history`** command.

See *IBM DB2 UDB Data Recovery and High Availability Guide and Reference*, SC09-4831 for further details.

4.2.2 Performance considerations

The performance of a backup operation depends upon a number of factors including the volume of data, size and number of available buffers, parallelization, number and speed of the media, and *throttling*[1].

Backup tends to be I/O-bound, especially the target devices. The goal should be to achieve full target device utilization, especially for tape devices where stalling during I/O is very expensive. In general, reading from the database is much faster than writing to the backup devices.

In general, the smaller the volume of data, the greater the size and number of buffers, the exploitation of parallel execution, the use of multiple high speed target devices, and specification of minimal throttling will yield superior performance. However, this requires adequate system resources (CPU, I/O, and memory) to be made available for backup, which may not be viable in some environments.

Note: The **`util_heap_sz`** database configuration parameter specifies the maximum database shared memory available for use by the **`backup`**, **`restore`** and **`load`** utilities. An insufficient **`util_heap_sz`** specification may prevent concurrent execution of these utilities.

4.2.3 Best practices

We recommend the following best practices to achieve superior backup performance:

[1] Throttling is a new feature in DB2 UDB V8 FP1 that enables you to limit a backup process from consuming more than a certain percentage of available non-idle computing resources, so that the performance impact on other concurrently running processes can be controlled.

1. Reduce the volume of data to be backed up.

   In general, the volume of data involved depends upon whether the granularity of the backup is the entire database, or just some of the table spaces.

   Table space-level backups tend to involve smaller volumes of data, and also facilitate the management of table data, indexes, and long field data or large objects (LOB) data in separate table spaces. However, the decision to choose between a database or table space-level backup is dependent on factors besides performance (such as availability), and is therefore not discussed here.

   Another backup option that may further reduce the volume of data to be backed up is the choice of incremental or delta backups. The decision to choose incremental/delta backups should be based on the percentage of changed data relative to the total volume of data in the database or table space. The smaller the percentage, the smaller is the volume of data to be backed up, and therefore the better the performer.

2. Increase the backup buffer size.

   The **BUFFER** buffer-size parameter specifies the size in 4 KB pages to be used in building the backup image. The default value is 1024, which is 4 MB.

   – For table spaces, the backup buffer size should be a multiple of the table space extent size +1. If multiple table spaces with different extent sizes are being backed up, then choose a value that is a multiple of the largest extent size.

     ```
     buffer size = ((N x ((extent size) x (number of containers)) + 1)
     ```

     where N is an integer.

     The extent size of a table space can be determined by issuing the following command:

     ```
     db2 list tablespaces show detail
     ```

     In general, the larger the buffer size the better, except when the backup is to a tape device, in which case smaller buffers should be specified.

   Note: If the backup image is written to tape with a variable block size, reduce the buffer size to within the range that the tape device supports. Otherwise, even though the backup operation may succeed, the resulting image may not be recoverable. The SQL2058W message is generated when the block size is larger than the block size supported by the tape device.

3. Increase the number of backup buffers

Incrementally increase the number of buffers until no performance improvements are observed (this is a trial and error exercise).

Note: If multiple I/O channels are available for the backup image, choose at least twice the number of buffers as channels to ensure that the channels do not wait for data.

4. Increase the value of the PARALLELISM parameter.

   This parameter defines the number of processes (UNIX) or threads (Windows) that are started when reading data from the database (the default is 1).

   Each process or thread is assigned to a specific table space. When it finishes backing up this table space, it requests another. Each process or thread incurs both memory and CPU overhead.

   Choose a value that corresponds to the number of table spaces being backed up as well as the number of CPUs.

   When there are a very large number of table spaces involved that exceed the number of backup target devices, then having a parallelism value greater than the number of target devices is only beneficial when the table space containers are highly utilized such that they are individually slower than the backup target devices. Typically, parallelism should be less than or equal to the number of target devices.

   Note: In a resource-constrained or heavily loaded system, consider staying with the default value of 1.

5. Use multiple high speed target devices.

   The backup image can made to span multiple targets using the **`TO dir/dev`** parameter. These targets will be written to in parallel, thereby speeding up the backup operation. It is preferable to initially back up to high speed disks, and later move the backups to tape at the earliest convenient time.

6. No throttling.

   The default is no throttling.

4.3 Export

The **`export`** command exports data from a DB2 database to one of several external formats, for subsequent import into other databases.

The following subsections are organized as follows:

- Brief description
- Performance considerations
- Best practices

4.3.1 Brief description

The **export** command is used to write data from a DB2 database to one or more files stored outside of the database. The exported data can then be imported or loaded into the same or different DB2 database (using the DB2 **import** or the DB2 **load** utility, respectively), or it can be imported into another application (for example, a spreadsheet).

The data to be exported can be specified by a SELECT statement, or by providing hierarchical information for typed tables.

Following is an example of the **export** command:

```
db2 export to staff.ixf of ixf select * from userid.staff
```

Note: When LOBs are involved, multiple paths may be provided to store LOB data to avoid out-of-space conditions in the file system.

The **export** utility is an embedded SQL application and does SQL fetches internally.

See *IBM DB2 UDB Data Movement Utilities Guide and Reference*, SC09-4830 for further details.

4.3.2 Performance considerations

The following performance considerations apply:

- Since the **export** utility does SQL fetches internally, all optimizations available to SQL operations apply to **export** as well, such as large buffer pools, indexing, and sort heaps.
- Minimize device contention on the output files.

4.3.3 Best practices

We recommend the following best practices to achieve superior **export** performance.

1. Ensure that all considerations applicable to tuning dynamic SQL statements are in place for the SQL used in the **`export`** command. Refer to 3.3.6, "Writing efficient SQL" on page 155 for SQL tuning for specific details.
2. Place the output files away from the containers and log devices in order to minimize contention.

4.4 Import

The **`import`** command inserts data from an input file generated by the **`export`** command into a table, hierarchy, or updateable view.

The following subsections are organized in this way:

- Brief description
- Performance considerations
- Best practices

4.4.1 Brief description

The **`import`** utility may either append to or replace the data in the target table/view.

The **`import`** utility is an embedded SQL application and does SQL inserts internally.

See *IBM DB2 UDB Data Movement Utilities Guide and Reference*, SC09-4830 for further details.

4.4.2 Performance considerations

The following performance considerations apply:

- Since the **`import`** utility does SQL inserts internally, all optimizations available to SQL inserts apply to import as well, such as large buffer pools and block buffering.
- By default, automatic commits are not performed, and **`import`** will issue a commit at the end of a successful import. While fewer commits improve overall performance in terms of CPU and elapsed time, they can negatively impact concurrency and restartability of **`import`** in the event of failure. In the case of a mass import, log space consumption could also become an issue and result in log full conditions, in some cases.

  The **`COMMITCOUNT n`** parameter specifies that a commit should be performed after every n records are imported. The default value is zero.

- By default, **import** inserts one row at a time into a target block and checks for the return code. This is less efficient that inserting a block at a time.

  The **MODIFIED BY COMPOUND = x** parameter (where x is a number between 1 and 100, inclusive) uses non-atomic compound SQL to insert the data, and x statements will be attempted each time. The **import** command will wait for the SQL return code about the result of the inserts after x rows instead of the default one row. If this modifier is specified, and the transaction log is not sufficiently large, the import operation will fail.

  The transaction log must be large enough to accommodate either the number of rows specified by **COMMITCOUNT**, or the number of rows in the data file if **COMMITCOUNT** is not specified. It is therefore generally recommended to use **COMMITCOUNT** along with **COMPOUND** in order to avoid transaction log overflows.

- **load** is generally faster than **import**, but it does not support loading data at the hierarchy level. The **import** utility has other advantages such as firing triggers, clustered index support and concurrent maintenance of constraints, materialized query tables and referential integrity.

  See Appendix B, “Differences between the Import and Load Utility” in *IBM DB2 UDB Data Movement Utilities Guide and Reference*, SC09-4830, for further details.

4.4.3 Best practices

We recommend the following best practices to achieve superior **import** performance.

1. Ensure that all considerations applicable to tuning SQL inserts are applied including the specification of very large buffer pools and buffered inserts. Refer to 3.3.6, “Writing efficient SQL” on page 155 for SQL tuning for specific details.
2. In a partitioned database environment, enable the **import** utility to use buffered inserts. This reduces the messaging that occurs when data is imported, resulting in better performance. However, since details about a failed buffered insert are not returned, this option should only be enabled if one is not concerned about error reporting.

   Use the DB2 **bind** utility to request buffered inserts. The **import** package, db2uimpm.bnd, must be rebound against the database using the INSERT BUF option.

   For example:

   ```
   db2 connect to your_database
   db2 bind db2uimpm.bnd insert buf
   ```

4.5 Load

The `load` utility moves data from files, named pipes, devices, or a cursor into a DB2 table.

The following subsections are organized in this way:

- Brief description
- Performance considerations
- Best practices

4.5.1 Brief description

The `load` utility is capable of efficiently moving large quantities of data into newly created tables, or into tables that already create contain data. If the table receiving the new data already contains data, you can replace or append to the existing data. The `load` utility can handle most data types, including large objects (LOBs) and user-defined data types (UDTs).

Note: The load utility does not fire triggers, and does not perform referential or table constraints checking; instead, it only validates the uniqueness constraint of indexes.

The `SET INTEGRITY` statement has to be executed against the table to validate referential or table constraints.

The `load` process consists of four distinct phases, as follows:

1. **Load phase** during which the data is written to the table. During this phase, data is loaded into the table, and index keys and table statistics are collected, if necessary. Save points, or points of consistency, are established at intervals specified through the `SAVECOUNT` parameter in the `LOAD` command. Messages are generated, indicating how many input rows were successfully loaded at the time of the save point.

   While the load operation is taking place, the target table is in the “load in progress” state. If the table has constraints, the table will also be in the “check pending” state. If the `ALLOW READ ACCESS` option was specified, the table will also be in the “read access only” state.

2. **Build phase** during which indexes are produced. During the build phase, indexes are produced based on the index keys collected during the load phase. The index keys are sorted during the load phase, and index statistics are collected (if the `STATISTICS YES` with `INDEXES` option was specified). The statistics are similar to those collected through the `RUNSTATS` command. If a

failure occurs during the build phase, the **RESTART** option automatically restarts the load operation at the appropriate point.

3. **Delete phase** during which the rows that caused a unique key violation or a DATALINK violation are removed from the table. Unique key violations are placed into the exception table (if one was specified), and messages about rejected rows are written to the message file. Following the completion of the load process, these messages should be reviewed for problems—and any problems found should be resolved—before re-inserting the corrected rows back into the table.
4. **Index copy phase** during which index data is copied from a system temporary table space to the original table space. This will only occur if a system temporary table space was specified for index creation during the load operation with the **READ ACCESS** option specified.

Note: The **load** utility is usually faster than the **import** utility because it writes formatted pages directly into the database, while the import utility performs SQL **INSERT**s.

See *IBM DB2 UDB Data Movement Utilities Guide and Reference*, SC09-4830*,* for further details.

4.5.2 Performance considerations

The performance of the **load** utility depends on the nature and the quantity of the data, the number of indexes, and the load options specified such as **ANYORDER, BINARY NUMERICS** and **PACKED DECIMAL, COPY YES** or **NO, CPU_PARALLELISM, DATA BUFFER, DISK_PARALLELISM, FASTPARSE, NONRECOVERABLE, NOROWWARNINGS, ALLOW READ ACCESS, SAVECOUNT, STATISTICS YES, USE <tablespace>**, and **WARNINGCOUNT**.

The following performance considerations apply:

- The **load** utility takes advantage of a hardware configuration in which multiple processors or multiple storage devices are used, such as in a symmetric multiprocessor (SMP) environment.

  There are several ways in which parallel processing of large amounts of data can take place using the **load** utility, as follows:

  – One way is through the use of multiple storage devices, which allows for I/O parallelism during the load operation. The **load** utility **DISK_PARALLELISM** parameter specifies the number of processes or threads used to parse, convert, and format data records.

  - Another way involves the use of multiple processors in an SMP environment, which allows for intra-partition parallelism. Both can be used together to provide even faster loading of data; the **`load`** utility **`CPU_PARALLELISM`** parameter specifies the number of processes or threads used to write data records to disk.

- Index creation performance depends the efficiency of the sort process which is in turn controlled by the sort heap (database configuration **`sortheap`** parameter), size of the buffer pool for the temporary table space, and the distribution of the temporary table space across multiple containers. Sort performance considerations are discussed 3.4.9, “Sort considerations” on page 275.
- Load performance can be improved by installing high performance sorting libraries from third party vendors to create indexes during load.
- The amount of data buffers (specified via the **`load`** utility **`DATA BUFFER`** parameter) allocated can have a significant impact on the I/O waiting time performance of the load utility especially when LOBs are involved. These buffers are allocated from the utility heap as specified by the database configuration **`util_heap_sz`** parameter.
- The **`load`** utility optionally supports concurrent indexing and statistics collection during the load process via the **`COPY YES`** or **`NO`** and **`STATISTICS YES`** parameters, respectively. Specifying these options will slow down the performance of the load process (including the potential impact on concurrency in case the **`load`** utility **`ALLOW READ ACCESS`** parameter is not chosen), but it needs to be evaluated in the context of overall performance should the load be serialized with a execution of the **`backup`** and **`runstats`** utilities.
- The individual **`load`** utility options can be tuned for superior performance provided their invocation does not compromise the functionality desired of the load process. These options are discussed in “Best practices” on page 308.

4.5.3 Best practices

We discuss best practices for achieving superior **`load`** performance from the following two perspectives:

- **General best practices** which apply regardless of the specific activity involved.
- **Task-specific best practices** for each specific activity such as performing combining multistep operations, building indexes, or loading an MDC table.

General best practices

The following general best practices are recommended:

1. Consider allowing the load utility to choose values for **DISK_PARALELLISM, CPU_PARALLELISM**, and **DATA BUFFER** parameters before attempting to tune them for your particular needs. The utility attempts to deliver the best performance possible by determining optimal values for these parameters based on the size and free space available in the utility heap.
2. Consider using the **TEMPFILES PATH** parameter for the location of the temporary files, if better I/O can be performed in another directory. The default is the database path.
3. Index creation performance is impacted by the database manager configuration parameter **sheapthres**, and the database configuration parameter **sortheap**, as well as the number and placement of the temporary table space containers. Performance considerations relating to these factors are discussed in 3.4.9, "Sort considerations" on page 275.

   Consider installing high performance sorting libraries from third party vendors to create indexes during the load operation. The **DB2SORT** environment variable (registry value) specifies the location of the sorting library to be used at run time.
4. The following load utility options may be tuned for superior performance in specific scenarios:
   - **ANYORDER**

     Specifying this file type modifier suspends the preservation of order in the data being loaded, and improves performance. This only applies when **INTRA_PARALLEL** is set to **YES**.

     Note: If the data to be loaded is presorted, the **ANYORDER** may not preserve the presorted order, and thereby negatively impact the performance of some queries.

   - **BINARY NUMERICS** and **PACKED DECIMAL**

     These file modifiers improve load performance when loading positional numeric ASC data into fixed-length records.
   - **COPY YES** or **NO**

     This parameter specifies whether or not a copy of the input data should be made during the load operation.

     - **COPY YES** reduces load performance, because of the increased I/O activity on an I/O-bound system as the input data is copied during the load operation. Distributing the I/O across multiple devices or directories on different disks can offset some of the performance penalty resulting from this operation.

Note that forward recovery (database configuration parameters **LOGRETAIN RECOVERY** or **USREXIT YES**) must be enabled.

- **COPY NO** results in the table being placed in backup pending state, therefore requiring the database, or selected table spaces, to be backed up before the table can be accessed. This may reduce the overall performance of a table that needs to be loaded and backed up before being used.

– **CPU_PARALLELISM**

This parameter can significantly improve load performance by exploiting intra-partition parallelism. It specifies the number of processes or threads used by the **load** utility to parse, convert, and format data records. The maximum number allowed is 30.

If there is insufficient memory to support the specified value, the utility adjusts the value accordingly.

Note: If this parameter is not specified, the load utility selects a default value that is based on the number of CPUs on the system.

The order of data in the input source is preserved regardless of the value of this parameter.

We recommend staying with the default.

Restriction: Although use of this parameter is not restricted to symmetric multiprocessor (SMP) hardware, it is unlikely that any discernible performance benefit will be observed in a non-SMP environments.

– **DATA BUFFER**

This parameter specifies the total amount of memory allocated to the load utility as a buffer. The **DATA BUFFER** parameter is useful when working with large objects (LOBs), and reduces I/O waiting time. The data buffer is allocated from the utility heap (**util_heap_sz** database configuration parameter whose default value is 5000 4 KB pages).

We recommend the following for the data buffer and utility heap:

- Choose the data buffer to be several extents in size.
- Because **load** is only one of several utilities that use memory from the utility heap, it is recommended that *no more than* fifty percent of the pages defined by this parameter be available for the **load** utility.

  - Depending on the amount of storage available on your system, you should consider allocating more memory for use by the DB2 utilities.

– **DISK_PARALLELISM**

This parameter can significantly improve load performance by exploiting parallel I/O against available containers. It specifies the number of processes or threads used by the **load** utility to write data records to disk. The maximum number allowed is the greater of four times the **CPU_PARALLELISM** value actually used by the **load** utility, or 50.

By default, **DISK_PARALLELISM** is equal to the sum of the table space containers on all table spaces containing objects for the table being loaded, except where this value exceeds the maximum number allowed.

We recommend staying with the default.

– **FASTPARSE**

This file type modifier can enhance performance by reducing the data checking that is performed on user-supplied column values.

Note: This option should only be used when the data being loaded is known to be valid.

– **NONRECOVERABLE**

This parameter will make the data/transactions loaded non-recoverable. **Load** performance is enhanced, because no additional activity beyond the movement of data into the table is required. At the end of the **load** operation, the table spaces are *not* in backup pending state.

Note: When these non-recoverable transactions are encountered during a subsequent restore and rollforward recovery operation, the table is *not* updated, and is marked “invalid”. Further actions against this table are ignored. After the rollforward operation is complete, the table can either be dropped or a LOAD TERMINATE command can be issued to bring it back online.

– **NOROWWARNINGS**

This file type modifier suppresses the recording of warnings about rejected rows, and can enhance performance when a large number of warnings are anticipated.

– **SAVECOUNT**

This parameter sets an interval for the establishment of consistency points during a **load** operation. A **LOAD RESTART** operation will automatically continue from the last consistency point. The synchronization of activities performed to establish a consistency point takes time. If done too frequently, there will be a noticeable reduction in load performance.

We recommend that when a large number of rows are to be loaded, a large **SAVECOUNT** value be specified, for example, a value of ten million when 100 million rows are being loaded.

– **STATISTICS YES**

This parameter specifies that data distribution and index statistics be collected during the **load** operation.

While the **load** operation is slower with this option (particularly when **DETAILED INDEXES ALL** is specified), the overall performance of a single step load with statistics gathering is more efficient than a multi-step operation involving a **load** operation followed by a **runstats** operation.

We recommend that when loading data into large tables, a larger value be chosen for the database configuration statistics heap size parameter **stat_heap_sz**. This parameter specifies the maximum size of the heap used in collecting statistics using the **runstats** command.

Note: The default value of **stat_heap_sz** is appropriate when no distribution statistics are collected (not the case when **STATISTICS YES** is chosen), or when distribution statistics are only being collected for relatively narrow tables.

– **USE <tablespaceName>**

This parameter allows an index to be rebuilt in a system temporary table space and copied back to the index table space during the index copy phase of a **load** operation. When a **load** operation in **ALLOW READ ACCESS** mode fully rebuilds the indexes, the new indexes are built as a shadow. The original indexes are replaced by the new indexes at the end of the load operation.

Note: The **USE <tablespaceName>** option is only supported for the **INDEXING MODE REBUILD** or **INDEXING MODE AUTOSELECT** options. If the **INDEXING MODE AUTOSELECT** option is specified and the load utility selects incremental maintenance of the indexes, the **USE <tablespaceName>** option will be ignored.

The **USE <tablespaceName>** option is ignored if the load operation is not in **ALLOW READ ACCESS** mode, or if the indexing mode is incompatible.

By default, the shadow index is built in the same table space as the original index. If the shadow index is built in the same table space as the original index, the original index will be instantaneously replaced by the shadow. Since both the original index and the new index are maintained simultaneously, there must be sufficient table space to hold both indexes at the same time. If the load operation is aborted, the extra space used to build the new index is released. If the load operation commits, the space used for the original index is released and the new index becomes the current index. When the new indexes are built in the same table space as the original indexes, replacing the original indexes will take place almost instantaneously.

Note: If the indexes are built in a DMS table space, the new shadow index cannot be seen by the user. If the indexes are built within an SMS table space, the user may see index files in the table space directory with the .IN1 suffix and the .INX suffix. These suffixes do not indicate which is the original index and which is the shadow index

However, if the shadow index is built in a system temporary table space, the load operation will require an index copy phase, which will copy the index from a system temporary table space to the index table space. There will be considerable I/O involved in the copy.

Note: To make sure that there is sufficient space in the original index table space, space is allocated in the original table space during the build phase. Therefore, if the `load` operation is going to run out of index space, it will do it during the build phase. If this happens, the original index will not be lost.

We recommend staying with the default.

– **`WARNINGCOUNT`**

This parameter specifies the limit on the number of warnings returned by the utility before the `load` operation is forced to terminate.

This parameter does *not* have a direct impact on load performance, but it allows you to terminate a `load` operation when an unexpectedly large number of warnings are encountered, thus avoiding having to wait until the end of the load to operation to become aware of the problem.

We recommend that you set this value to the approximate number of warnings expected when only a few warnings are expected (twenty, when no warnings are expected). This gives you the opportunity to correct data (or to drop and then recreate the table being loaded) before attempting to complete the `load` operation.

Task-specific best practices

We recommend the following best practices for:

- Combining multistep operations
- Building indexes
- Loading an MDC table

Combining multistep operations

As described earlier, the `load` utility supports concurrent execution of other operations such as taking a backup copy using the COPY YES or NO parameter, and collecting statistics using the STATISTICS YES parameter. You could also consider creation of indexes during load in the same category, since you have the choice of loading data in a step separate from index creation.

In general, the adoption of other concurrent operations during the `load` execution tends to slow down `load` performance. However, you need to compare this `load` performance with achieving the same requirements via a multistep operation involving `load` followed by a `backup` and/or `runstats` operation. With such a comparison, the concurrent load plus operations always outperforms the performance of multistep operations.

We therefore recommend that if permitted, concurrent operations be performed during load, after ensuring that adequate resources are provided to enhance the performance of concurrent operations, such as allocating adequate `stat_heap_sz`, `util_heap_sz, sortheap, sortheapthres` and buffer pool for the system temporary table space. In summary:

- Specify `COPY YES`.
- Specify `STATISTICS YES`.
- Create indexes during the load operation rather than via a multistep operation involving a load followed by `CREATE INDEX`.

Building indexes

Indexes are built during the build phase of a load operation. There are four indexing modes that can be specified in the `LOAD` command:

1. `REBUILD` will result in all indexes being rebuilt.
2. `INCREMENTAL` will cause all indexes to be extended with new data.
3. `AUTOSELECT` allows the `load` utility to automatically decide between `REBUILD` or `INCREMENTAL` mode (this is the default).
4. `DEFERRED` specifies that the `load` utility should not attempt index creation; indexes will be marked as needing a refresh, and a rebuild may be forced the first time they are accessed.

Note: This option is not compatible with the `ALLOW READ ACCESS` option, because it does not maintain the indexes—and index scanners require a valid index.

The main performance considerations are summarized as follows:

- Index creation performance can be significantly impacted by the amount of memory dedicated to the sorting of index keys during the load operation as specified by the database configuration parameter **`sortheap`**. If sort overflows cannot be avoided, then it is important that the buffer pool for the system temporary table spaces to be large enough to minimize the amount of disk I/O that overflows case.

  Furthermore, to achieve I/O parallelism during the merging[2] of sort runs, it is recommended that temporary table spaces be declared with multiple containers, each residing on a different disk drive. 3.4.9, "Sort considerations" on page 275 describes sort considerations in greater detail.

  The use of high performance sorting libraries from third party vendors to create indexes during the **`load`** operation can also enhance performance; the **`DB2SORT`** environment variable (registry value) specifies the location of the sorting library to be loaded at run time.
- In most cases, it is more efficient to update the indexes during the load operation than to invoke the `CREATE INDEX` statement for each index after the load. See "Combining multistep operations" on page 319 for further details.
- Creating shadow indexes in a separate temporary table space saves space in the table space containing the index, but it incurs the overhead of the index copy phase, which may not be acceptable.

Loading an MDC table

MDC tables are supported by a new physical structure that combines data, special kinds of indexes, and a block map. Therefore, MDC load operations will always have a build phase since all MDC tables have block indexes.

During the load phase, extra logging (approximately two extra log records per extent allocated) for the maintenance of the block map is performed. A system temporary table with an index is used to load data into an MDC tables. The size of the system temporary table is proportional to the number of distinct cells loaded. The size of each row in the table is proportional to the size of the MDC dimension key.

[2] If an index is so large that it cannot be sorted in memory, a sort overflow occurs. That is, the data is divided among several "sort runs" and stored in a temporary table space that will be merged later.

We recommend the following to enhance the performance of loading an MDC table:

1. Consider increasing the database configuration parameter **`logbufsz`** to a value that takes into account the additional logging for the maintenance of the block map.
2. Ensure that the buffer pool for the temporary table space is large enough in order to minimize I/O against the system temporary table.
3. Increase the size of the database configuration parameter **`util_heap_sz`** by 10-15% more than usual in order to reduce disk I/O during the clustering of data that is performed during the load phase.
4. When the **`DATA BUFFER`** option of **`load`** command is specified, its value should also be increased by 10-15%. If the **`load`** command is being used to load several MDC tables concurrently, the **`util_heap_sz`** database configuration parameter should be increased accordingly.

4.6 Reorg

A **`reorg`** can be used to improve clustering, regain free space, and eliminate overflow rows.

The following subsections are organized in this way:

- Brief description
- Performance considerations
- Best practices

4.6.1 Brief description

The DB2 **`reorg`** utility is used to reorder the physical layout of data within DB2 tables and indexes. It also reclaims the space of deleted keys in type 2 indexes.

Note: DB2 supplies the **`reorgchk`** command to assist in determining whether tables and/or indexes need to be reorganized. **`reorgchk`** allows you to use the current catalog statistics (last updated when the last RUNSTATS was run), or **`reorgchk`** can invoke **`runstats`** as part of **`reorgchk`** processing.

If the name of an index is specified as part of the **`reorg`** command, the database manager reorganizes the data according to the order in the index[3]. If the name of an index is not specified, and if a clustering index exists, the data will be ordered

[3] Typically the index chosen is the one that is most often used in SQL queries.

according to the clustering index. If you do not specify the name of an index and no clustering index exists, the records are reorganized without regard to order.

See *IBM DB2 UDB Command Reference*, SC09-4828, for further details.

4.6.2 Performance considerations

The performance of **reorg** depends upon the volume of data, number of indexes involved, and the availability of space within the table space holding the table being reorganized.

The following performance considerations apply:

- When a number of indexes are associated with a table being reorganized, the amount of memory available to sort the index keys will have a significant impact on performance.
- If there is not enough space within the table space to hold the table being reorganized, then a separate temporary table space needs to be specified.
- Parallelism of the index recreation can significantly impact the performance of **reorg**.
- A table that has **LOB** or **LONG** data types can have a significant negative impact on **reorg** performance.
- Reorganizing via a low clusterratio clustering index can be very slow since it is chosen as the access path to the data, and significant random I/Os to the data may occur.

4.6.3 Best practices

We recommend the following best practices to achieve superior **reorg** performance:

1. Ensure that adequate memory is available for sorting by adjusting the database manager **sortheapthres** parameter, and the database configuration **sortheap** parameter. Additionally, the buffer pool for the temporary table space should be made large enough to minimize I/O to disk in case sort overflows can not be avoided. Furthermore, to achieve I/O parallelism during the merging of sort runs, it is recommended that temporary table spaces be declared with multiple containers, each residing on a different disk drive. 3.4.9, “Sort considerations” on page 275 describes sort considerations in greater detail.
2. **reorg** can benefit from parallelism enabled via the database manager configuration **INTRA_PARALLEL** parameter—but you need to fully understand the DB2 instance-wide repercussions of resource consumption when this

parameter is enabled. This is not recommended in resource-constrained environments.

3. For tables with **LOB** or **LONG** data types, consider setting the DB2 registry variable **DB2REORGDATAONLY=ON**. This will bypass the reorganizing of **LOB** or **LONG** data.

   Note: A drawback of using **DB2REORGDATAONLY** would be that free space from deleted entries contained within the **LOB** or **LONG** table space would not be reclaimed.

4. Consider reorganizing the table more frequently to avoid **reorg** performance degradation due to low clusterratio clustering index access. In fact, the clusterratio of the clustering index (if one is present) should be monitored closely and reorganized promptly when slippage occurs below 95% in order to ensure the performance of SQL queries that would benefit from this access path.

4.7 Restore

The **restore** command is used in conjunction with the **backup** command to ensure recoverability of a database and/or table space.

The following subsections are organized in this way:

- Brief description
- Performance considerations
- Best practices

4.7.1 Brief description

The **restore** command is used to bring back a former database image that was backed up using the **backup** command. The restore can be at the database or table space level.

See *IBM DB2 UDB Data Recovery and High Availability Guide and Reference*, SC09-4831, for further details.

4.7.2 Performance considerations

The performance of a restore operation depends upon the size and number of available restore buffers, parallelization, and throttling.

In general, the greater the size and number of restore buffers, the exploitation of parallel execution and the specification of minimal throttling will yield superior performance. However, this requires adequate system resources (CPU, I/O and memory) to be made available for backup, which may not be viable in some environments.

Note: The **`util_heap_sz`** database configuration parameter specifies the maximum database shared memory available for use by the **`backup`**, **`restore`** and **`load`** utilities. An insufficient **`util_heap_sz`** specification may prevent concurrent execution of these utilities.

4.7.3 Best practices

We recommend the following best practices to achieve superior restore performance; these are very similar to the recommendations made for the **`backup`** command.

1. Increase the restore buffer size

   The **`BUFFER`** buffer-size parameter specifies the size in 4 KB pages to be used in restoring the backup image (the default value is 1024).

   The restore buffer size should be an integer multiple of the backup buffer size specified during the backup operation.

   Note: If an incorrect buffer size is specified, the buffers allocated will be the smallest acceptable size.

   To determine the buffer size used in creating the backup image, issue the following command:

   ```
   db2ckbkp -h <filename of backup image>
   ```

   Example 4-1 shows the sample output of such a command.

Example 4-1 Determining backup buffer size

```
db2ckbkp -h SAMPLE2.0.krodger.NODE0000.CATN0000.19990818122909.001
=====================
MEDIA HEADER REACHED:
=====================
Server Database Name --SAMPLE2
Server Database Alias --SAMPLE2
Client Database Alias --SAMPLE2
Timestamp --19990818122909
Database Partition Number --0
Instance --krodger
Sequence Number --1
Release ID --900
Database Seed --65E0B395
DB Comment ’s Codepage (Volume)--0
```

```
DB Comment (Volume)--
DB Comment 's Codepage (System)--0
DB Comment (System)--
Authentication Value --255
Backup Mode --0
Backup Type --0
Backup Gran.--0
Status Flags --11
System Cats inc --1
Catalog Database Partition No.--0
DB Codeset --ISO8859-1
DB Territory --
Backup Buffer Size --4194304
Number of Sessions --1
Platform --0
The proper image file name would be:
SAMPLE2.0.krodger.NODE0000.CATN0000.19990818122909.001
[1 ]Buffers processed:####
Image Verification Complete -successful.
```

2. Increase the number of buffers.

   Incrementally increase the number of restore buffers until no performance improvements are observed (this is a trial and error exercise).

3. Increase the value of the **PARALLELISM** parameter.

   This parameter defines the number of buffer manipulators (BM) that will be used to write to the database/table space during the restore operation. The default is 1.

   Each BM is assigned to a specific table space. When it finishes restoring this table space, it requests another. Each BM incurs both memory and CPU overhead.

   During restore, it is reasonable to assume that pages can be written to table space containers faster than they can be read from the backup images sources. As long as this assumption is true, then a parallelism value that is less than the number of sources may cause the backup image I/O to halt temporarily (a parallelism value that is greater than the number of backup image sources offers diminishing returns at some small cost). Choose a parallelism value equal to the number of backup image sources.

4. No throttling

   The default is no throttling.

4.8 Runstats

The **runstats** command collects statistics about the physical characteristics of a table and its associated indexes and records them in the system catalog. These characteristics include the number of records, number of pages, average record length, and data distribution statistics.

The following subsections are organized as follows:

- Brief description
- Performance considerations
- Best practices

4.8.1 Brief description

The **runstats** utility gathers statistics about data within DB2 tables and indexes, and these statistics are used by the DB2 optimizer to generate optimal query access plans.

The following key options impact the performance of the **runstats** utility, but provide detailed statistics of significant benefit to the DB2 optimizer in its access path selection:

- **WITH DISTRIBUTION** clause
- **DETAILED** clause
- **LIKE STATISTICS** clause

WITH DISTRIBUTION clause

The **runstats** utility, by default, collects information about the size of the table, the highest and lowest values in the index(es), the degree of clustering of the table to any of its indexes, and the number of distinct values in indexed columns. However, when the optional **WITH DISTRIBUTION** clause is specified, the **runstats** utility collects additional information about the distribution of values between the highest and lowest values, as well.

The DB2 optimizer can exploit this additional information to provide superior access paths to certain kinds of queries when the data in the table tends to be skewed.

DETAILED clause

The **runstats** utility also provides an optional **DETAILED** clause which collects statistics that 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.

This additional information helps the optimizer make better estimates of the cost of accessing a table through an index.

The SAMPLED option, when used with the DETAILED option, allows **`runstats`** to employ a CPU sampling technique when compiling the extended index statistics. If this option is not specified, every entry in the index is examined to compute the extended index statistics.

LIKE STATISTICS clause

This optional clause collects additional column statistics (**`SUB_COUNT`** and **`SUB_DELIM_LENGTH`** in **`SYSSTAT.COLUMNS`**) for string columns only.

This additional information helps the DB2 optimizer make better selectivity estimates for predicates of the type “**`column_name LIKE ‘%xyz`**’” and “**`column_name LIKE ‘%xyz%`**’”, and thereby generate a superior access path for the query.

4.8.2 Performance considerations

The performance of the **`runstats`** utility depends upon the volume of data, the number of indexes associated with it, and the degree of detailed information requested via the **`WITH DISTRIBUTION`** and **`DETAILED`** clauses.

The following performance considerations apply:

- The **`runstats`** utility collected statistical information is critical to the DB2 optimizer’s selection of an optimal access path, and it is therefore imperative that such information be kept up to date. However, **`runstats`** consumes significant CPU and memory resources and should only be executed when significant changes have occurred to the underlying data that impact current statistics information and consequently the selection of an optimal access path by the DB2 optimizer.

  This implies that the frequency of **`runstats`** execution should be managed.

- The degree of statistical detailed information requested has a direct impact on the performance of the **`runstats`** utility. Specifying the **`WITH DISTRIBUTION`** clause with some or all columns, and/or the **`DETAILED`** clause, results in significant CPU and memory consumption. In particular, the database configuration parameter **`stat_heap_sz`** should be adjusted to accommodate the collection of detailed statistics.

Consider using the SAMPLED option of the DETAILED clause to reduce CPU consumption—this is of particular benefit in BI environments.

4.8.3 Best practices

We recommend the following best practices to achieve superior **`runstats`** performance:

1. In spite of the overhead of collecting detailed statistics, consider specifying the **`WITH DISRIBUTION`** and **`DETAILED`** clause on a table, if it is a very critical table in critical queries, is relatively small, or does not change too much and there is not too much activity on the system itself.
2. If the time to collect statistics is limited, and the table is relatively large and/or changes a lot, it might be beneficial to execute **`runstats`** and limit it to the set of columns that are used in predicates. This enables you to run the utility more frequently.
3. The **`WITH DISTRIBUTION`** option should be used when the data is known to have non-uniform data distribution and the workload is capable of exploiting non-uniform data distribution statistics.

   Keeping distribution statistics is advisable if at least one column in the table has a highly “non-uniform” distribution of data and the column appears frequently in equality or range predicates.

   Note: 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.

4. If there are many indexes on the table and **`DETAILED`** (extended) information on the indexes might improve access plans, consider using the **`SAMPLED`** option to reduce the time it takes to collect statistics.

   Regardless of whether the **`SAMPLED`** option is used or not, collecting detailed statistics on indexes is time-consuming. The **`SAMPLING`** option, when used with the **`DETAILED`** option, allows **`runstats`** to employ a CPU sampling technique when compiling the extended index statistics. If the option is not specified, every entry in the index is examined to compute the extended index statistics.

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

5. **DETAILED** index statistics should be collected 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.

   Note: The **DETAILED** statistics **PAGE_FETCH_PAIRS** and **CLUSTERFACTOR** will be collected only if the table is of a sufficient size, around 25 pages.

6. If there is skew in certain columns and predicates of the type "**column_name = <constant>"**, then it may be beneficial to specify a larger **NUM_FREQVALUES** value for that column
7. For columns that have range predicates (for example, "**column_name >= <constant>**", or "**column_name BETWEEN <constant1> AND <constant2>**"), or of the type "**column_name LIKE '%xyz**'", then it may be beneficial to specify a larger **NUM_QUANTILES** value.
8. If storage space is a concern and you cannot afford too much time on collecting statistics, do not specify high **NUM_FREQVALUES** or **NUM_QUANTILES** values for columns that are not used in predicates.
9. Collect additional column statistics via the **LIKE STATISTICS** clause for tables with string data types that are likely to be the target of SQL applications using wildcard LIKE predicates, such as "**LIKE '%xyz**'" or "**LIKE '%xyz**'".

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

   ```
   column_name 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.

5 Operating system considerations

In this chapter we provide an overview of operating system considerations relevant to achieving superior DB2 performance. We review hardware, operating system, filesystems, and memory issues relevant to DB2 performance. We also describe some of the performance monitoring tools available.

The topics covered include:

- AIX platform
- Windows platform

5.1 Introduction

The objective of this chapter is to familiarize DBAs with operating system platform performance drivers that can have a critical impact on the performance of their DB2 environment, so that they may be in a position to negotiate more effectively with system administrators responsible for these performance drivers.

Important: In most cases, the DBA has no jurisdiction over monitoring and tuning these key performance drivers, since they are the responsibility of the appropriate administrator. This chapter is not meant to be a comprehensive discourse on monitoring and managing operating system platform performance drivers; readers are strongly advised to consult other documentation sources for more detailed information about this subject.

The platforms discussed are AIX and Windows.

5.2 AIX platform

AIX is the IBM implementation of the UNIX operating system, and is available on the IBM Power-based and RS64 microprocessor-based systems. It includes a logical volume manager, and supports dynamic logical partitions[1] on POWER4™-based systems.

We briefly discuss the following key performance drivers:

- Operating system considerations
- Memory considerations
- Disk and filesystem considerations

Best practices and recommendations are provided for each of these performance drivers, and are classified as being general and DB2-specific. Typical performance monitoring tools for these performance drivers are also described.

5.2.1 Operating system considerations

In this section, we provide the following:

1. AIX/DB2 review
2. General performance recommendations
3. DB2-specific performance recommendations

[1] The ability to alter the amount of system memory or number of processors.

AIX/DB2 review

DB2 V8 supports both AIX 4.3.3 and AIX 5 versions.

Attention: There are no significant performance benefits in using either operating system version when using 32-bit DB2 and standard filesystems.

Systems that use POWER3, POWER4 or RS64 microprocessors are capable of running both 32-bit and 64-bit operating systems, and applications such as 64-bit DB2 V8. Each operating system release CD set contains both 32-bit and 64-bit versions of AIX.

Note: 64-bit applications can run on the 32-bit operating system version and vice versa; you can install the 32-bit version of AIX 5 and run 64-bit DB2 V8 on it, or boot the 64-bit version of AIX and run 32-bit DB2.

There is a small performance benefit in having the operating system and DB2 use the same bit size.

eServer™ pSeries® systems with POWER4 processors can take advantage of the IBM logical partitioning (LPAR) technology, which provides several independent system images running on a single hardware platform. LPAR technology with AIX 5 enables the system administrator to configure varying number of CPUs, memory size, and I/O adapters for each partition.

AIX 5.2 provides dynamic LPAR (DLPAR), where the number of CPUs and amount of memory can be varied online without rebooting the operating system. Support for dynamic re-configuration of I/O adapters between partitions is expected to be available in a future release of AIX.

LPAR enables the DB2 environment to be reconfigured as workload changes over time or at specific intervals. For example, during heavy month-end processing, CPUs normally allocated to the development/test/regression environment could be temporarily allocated to the production environment.

General recommendations

We recommend the following general best practices:

1. Use AIX 5.2 if possible at the latest maintenance level.
2. Use 64-bit DB2 with the 64-bit AIX kernel.
3. Use the 32-bit kernel if you do not use 64-bit DB2.
4. As a rule of thumb, UNIX systems should be configured with enough resources (CPU, memory, and so on) so that steady state system utilization is

generally 70% or less. This provides spare capacity to cope with unexpected peak workloads.

5. When the AIX system boots, the number of licensed users is used to size various operating system data structures, such as the total number of UNIX processes the system can handle. Running out of these structures can cause the system to slow down.

   The number of licensed users should at least be equal to the expected maximum number of concurrent connects. Overconfiguring these data structures is not recommended, since they consume memory that could be better utilized elsewhere.

   This number of licensed users also limits the number of uses who can log in to the system concurrently using **`telnet`**, **`rlogin`** and so on (but excluding Web-connected sessions). The default value is 2, which needs to be changed after installing AIX.

   The following command displays the current value of this variable:

   ```
   lslicense
   ```

   You need to be the superuser to run this command.

   Note: With AIX 5.2, the number of licensed users can be increased dynamically without requiring a system reboot, if you use the “IMMEDIATELY update AVAILABLE FIXED licenses” option.

6. AIX has a default value of 128 for the number of processes per user.

   This value needs to be increased to avoid DB2 failures under workloads involving large numbers of users. This value needs to be greater than the total of all DB2 processes with the same user id, which is typically the same DB2 instance.

   This count should include the total number of listeners, agents and subagents, prefetchers, page cleaners and so on for each instance—and the largest of these totals of all instances should be used. Unlike the number of licensed users variable, no operating system resource costs are dependent on the value specified for this variable.

   The following command displays the current value of this variable:

   ```
   lsattr -El sys0 | grep maxuproc
   ```

7. DB2 uses the kernel’s asynchronous I/O feature whenever it reads or writes more than one extent in a single request.

   The kernel uses “aio” threads to perform this I/O, and the minimum and maximum numbers of threads needs to be configured.

If too many are configured, the threads will not be used; if too few are configured, performance degrades. While there are several rules of thumb for determining the correct numbers, we recommend the following:

- Set `maxAIOservers` to the maximum of:
  - Minimum of 10 times the number of database disks, or 10 times the number of CPUs in the system
  - At least one per page cleaner and prefetcher (per instance)
- A good value for `minAIOservers` is `maxAIOservers/2`.

The following command displays the current settings of these variables:

```
lsattr -El aio0
```

DB2-specific recommendations

We recommend the following DB2-specific best practices:

1. DB2 in 64-bit mode enables DB2 to use larger heaps, but the DB2 processes such as db2agent also use more memory. This additional memory (approximately 50 KBytes per process) needs to be factored into memory usage computations. However, in most situations, heap usage outweighs process memory usage and the overhead of 64-bit process memory usage is not a consideration.
2. If you run a DB2 32-bit application on a 64-bit database, you will need to rebind the application and set UNIX environment variables to point to the 32-bit libraries, for example:

   ```
   LIBPATH=<db2instance home>/sqllib/lib32
   export LIBPATH
   ```
3. Multiple versions of DB2 ESE can be installed on the same system. With DB2 V8.1 Fixpak 2, multiple fixpaks of the same DB2 version can also coexist on the same system.

5.2.2 Memory considerations

In this section, we provide the following:

1. A review of the AIX virtual memory architecture
2. General performance recommendations
3. DB2-specific recommendations

AIX virtual memory architecture review

Virtual memory provides a mechanism for the system to assign more memory to processes than physically exists on the system. It does this by moving (or paging) temporarily unused parts of memory (pages) to disk, and restoring them when required.

Figure 5-1 shows a simplified model of AIX memory architecture.

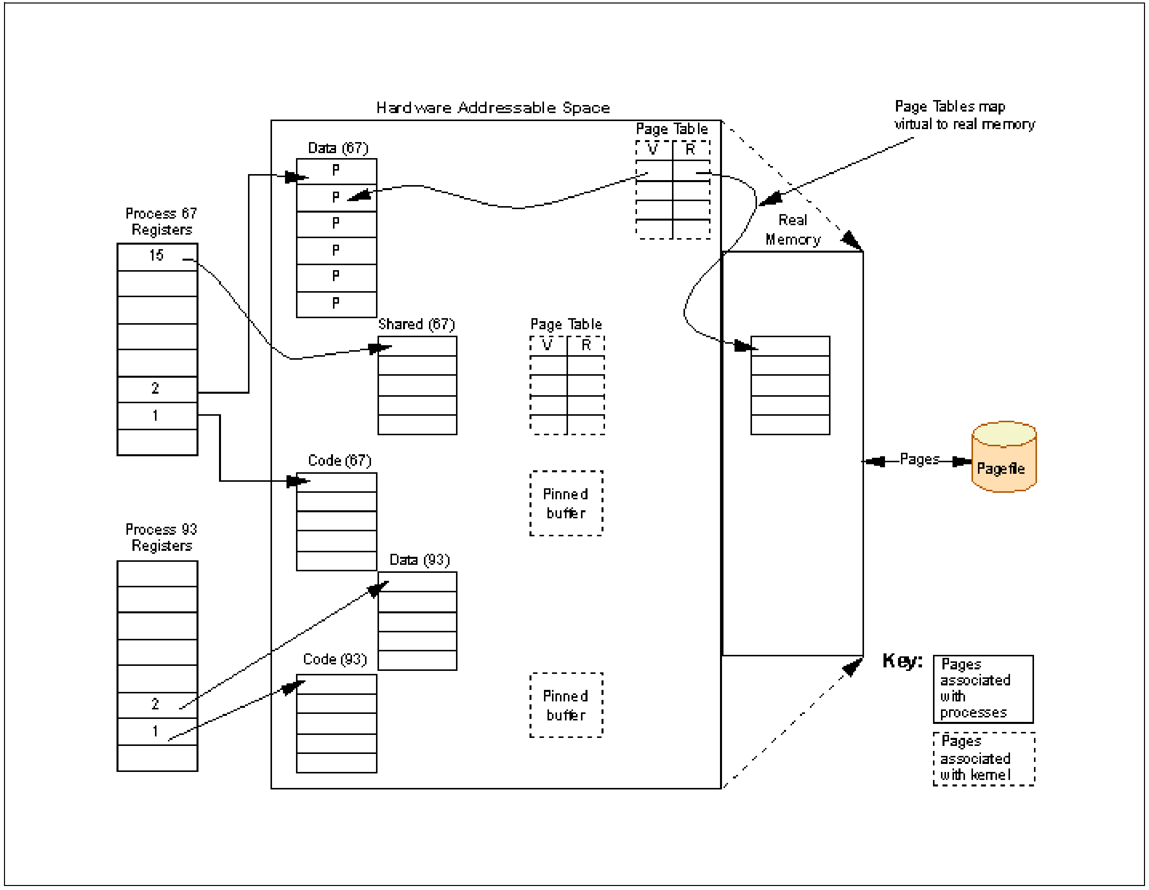


Figure 5-1 Simplified view of 32-bit AIX memory architecture

The main elements are real or physical memory, virtual memory addressable space, segment registers, page tables and page files.

- All systems have a certain amount of real memory; the maximum varies by processor model.
- Process virtual memory, on the other hand, is limited by the architecture: for 32-bit processes, the limit is 4 GB, while for 64-bit processes, the limit is approximately 2^{36}GB.

- All processes are associated with a set of segment registers, which is 16 in 32-bit environments, but a practically unlimited number in 64-bit environments.

  Processes use segment-based addresses to access memory segments, which can be up to 256 MB in size in 32-bit environments. Each segment consists of a number of pages of the same size; pages are typically 4 KB in size, but AIX 5 permits the creation of larger page sizes of 16 MB.
- Page tables are used to map virtual memory addresses to physical/real memory addresses. There is a page table for every process.
- Page files store virtual memory pages that have been paged out.

A brief overview of system view of memory, paging, paging space, and DB2 process segment register use is provided in the following subsection.

System view of memory

The operating system uses virtual memory for processes, as well as to hold data being read or written to files.

When a process reads from a file, the data is transferred from the filesystem into operating system buffers and then copied into the process's memory space. The operating system may read more data from the disk than the process actually requested; that is, it may "read ahead" so that a subsequent read by the process will be satisfied from these buffers instead of from disk. Similarly, when a process writes to a file, the data is transferred to operating system buffers, and then scheduled to be written to disk at a later point in time.

The operating system balances the amount of memory available for processes and buffers. Too much memory for processes will result in poor filesystem performance, while too much memory for filesystem buffers will result in processes paging excessively (see "Paging" on page 337), thereby degrading performance. This is tunable.

For systems with large amounts of filesystem I/O, the amount of memory for filesystem buffers should be large. For systems using mostly raw disks, reducing the memory for filesystem buffer pools will provide more memory for processes.

Note: For a more extensive discussion of this subject, refer to Chapter 3 in *The POWER4 Processor Introduction and Tuning Guide*, SG24-7041.

Paging

Whenever possible, the operating system will retain virtual memory pages in real memory to enable maximum reuse. If there is insufficient real memory to hold the required pages, the operating system has to perform paging.

When a process requests a page of virtual memory that has not been mapped to a real memory page, the process is said to have incurred a *page fault*. This involves a wait by the process until the requested virtual memory page is mapped to real memory. Page in is the task of making room in real memory to store currently requested virtual memory pages by stealing real memory space previously allocated to someone else.

- If the page to be discarded has been changed, it must first be written to disk; this task is called *page out*. If the page was part of a process's data or stack space, it is written to the page file (see "Paging space" on page 339). If it is part of a filesystem, it is written back to the filesystem. These I/Os correspond to paging and filesystem page outs, respectively.
- If the page has not been changed (such as a page that holds the process's code), then it is simply discarded because the code page can always be reread from the executable file itself.

Note: Some virtual memory pages cannot be paged out if they happen to be "pinned"; such virtual memory pages hold page tables and filesystem buffers while a disk transfer is in progress.

Once real memory space is freed, the operating system reads in the appropriate page from the page file or filesystem. A read from a filesystem file might bring in a page of data, or a page of a process's code.

The algorithm used to discard filesystem and process pages is loosely based on two operating system variables **`minperm`** and **`maxperm`** that represent percentages of memory, as follows:

- If the percentage of real memory occupied by file pages rises above **`maxperm`**, the operating system discards file pages.
- If the percentage of real memory occupied by file pages drops below **`minperm`**, the operating system discards both file and process pages.
- If the percentage of real memory occupied by file pages is between **`minperm`** and **`maxperm`**, the operating system discards file pages while the page fault rate[2] for file pages is greater than the page fault rate for process pages.

Note: Default value for **`minperm`** is 20%, while that of **`maxperm`** is 80%.

A number of commands, such as **`vmtune`** and **`nmon`** (see "Monitoring and problem determination tools" on page 387), display the current **`minperm`** and **`maxperm`**

[2] This refers to the rate at which the operating system is transferring pages to and from the paging space.

values. These tools will also display the value of **`numperm`**, the current percentage of memory devoted to file pages.

Paging space

Paging space on disk is required to hold virtual memory paged out by the operating system. This space is allocated by the UNIX sysadmin. The amount required will depend on a number of factors, including the following:

- Amount of physical memory on the system
- Number of processes
- Size of any heaps allocated

AIX implements several paging policies.

The default policy (which is recommended) is to only allocate paging space (pages in the page file) when a virtual memory page must be written to disk (known as *deferred paging*).

Note: Typically, for the default policy, the amount allocated to a paging file is between one and two times the amount of physical memory. We recommend paging space to be twice the amount of physical memory.

The **`lsps -s`** command shows the utilization as a percentage, as described in "lsps" on page 369.

Because of deferred paging, it is possible to see a condition in which paging space usage spikes very suddenly. If memory becomes overcommitted in the system by even a small amount, then pages begin to get paged out to disk, thereby consuming new paging space. Once particular pages of memory get backed by paging space, they continue to do so for the lifetime of the process which allocated them, *even if* those pages are later paged back into physical memory.

Therefore, as more and more memory gets paged out for the first time, paging space usage continues to climb. In extreme cases, this can happen very rapidly, leading to most or all memory held by agents on the system consuming space on the paging volume.

If you run out of paging space, the system will probably stop.

Note: When paging space is very low, AIX sends a SIGDANGER signal to all processes on the system; DB2 has a signal handler for this signal and will log an entry in the db2diag.log.

Despite the problem of running out of paging space, monitoring the page fault rate is even more important than monitoring the amount of paging space used, because high page fault rates correspond to large amounts of I/O and have a significant adverse performance impact. High page fault rates are typically in the range of hundreds or thousands of page faults per second, but this is subjective since it depends upon the number of CPUs in the system, as well as the application workload.

When DB2 agent processes finish processing a request and are returned to the agent pool, they do not release their agent private memory. This memory is kept for fast reuse to improve performance. However, it can result in increased memory and page space consumption.

This behavior can be changed by setting the DB2 registry variables **DB2MEMDISCLAIM** and **DB2MEMMAXFREE**.

- **DB2MEMDISCLAIM = YES** means the agent process discards some or all private memory back to the agent heap.
- **DB2MEMMAXFREE** determines how much memory each idle agent retains. The default value for **DB2MEMMAXFREE** is 8 MB.

The disclaim mechanism is important when the amount of paging space available is strictly limited.

Attention: The page fault rate should ideally be zero, but 10 or 20 page ins or page outs per second per CPU may be acceptable.

There are a number of tools which show the page fault rate. In AIX, page faults also include reads and writes to files in filesystems. Only some tools such as **nmon** distinguish these from process page faults, as shown in Figure 5-18 on page 372.

The **ps** command with the **v** option displays the number of page ins for a particular process, as shown in Figure 5-21 on page 375.

DB2 process segment register use

Processes on IBM pSeries platforms (Power or RS64 processors) access memory via segment-based addresses. A segment-based address is calculated using a segment register and a segment offset.

As indicated earlier, a 32-bit environment has 16 registers per process, where each register can reference a memory segment of up to 256 MB. If 32-bit programs such as DB2 or other user processes need to access more than 256 MB of data, they can do so by using a contiguous set of segment registers. A 64-bit environment supports an effectively unlimited number of segment registers. However, the method of determining addresses is identical in both environments.

Some segment registers are reserved for use by AIX or for DB2 internal operations, while others can be used for several purposes.

Note: By default, segment 2 holds process data. This includes any constants, non-stack, and stack variables. Constants and non-stack variables are allocated from the bottom of the segment 2 upwards, while stack space is allocated from the top of the segment 2 downwards.

Figure 5-2 shows the DB2 processes defaults for segment registers in the 32-bit environment. The maximum addressable space in a 32-bit environment by a process is 15 x 256 MB or approximately 3.75 GB, since segment register (0) is reserved for mapping kernel space.

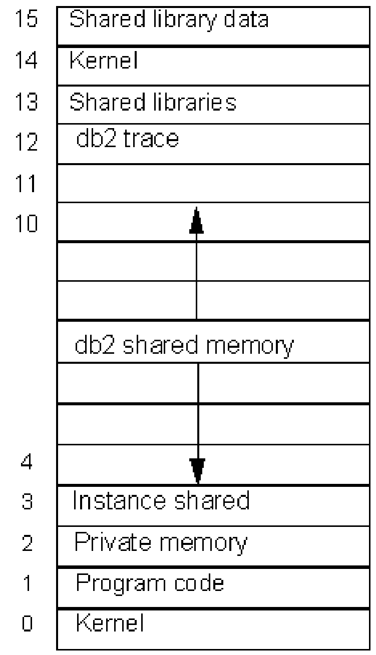


Figure 5-2 32-bit environment Segment Register usage

DB2 processes allocate virtual memory dynamically for private agent memory or DB2 shared memory. These are addressed through various segment registers as shown in Figure 5-2. Some of this memory may be shared with other DB2 processes, in which case each process's registers simply point to a common area of memory.

For 32-bit processes, some of the assignment of segment registers is as follows:

- Segment register 2 holds all of the DB2 private memory for an agent. This private memory includes all private data used for SQL processing, such as private sorts, local data for processing of SQL statements, and heaps which reside in agent private memory, as shown in Figure 2-7 on page 32.

  As described earlier, the process stack in this segment grows downwards from the upper bound address, while non-stack memory is allocated from the bottom upwards. These may collide if instance memory configuration parameters are overconfigured, resulting in unpredictable process failures.
- Segment register 3 is used to attach to the instance shared segment.
- Segment registers 4 through 11 are assigned *contiguously* to attach DB2 shared memory as needed. This includes buffer pools, database heap, utility heap, package cache, locklist, and sort heap threshold.

  The following are some of the main segments that will be needed for the following configuration parameters (this is not a comprehensive list):

  – Segment register 8 is used for FCM communications with Distributed Partitioning Feature (DPF) if the **`DB2_FORCE_FCM_BP`** registry variable is set to YES.

  – Segment register 9 is used for agent to UDF communication.

  – Segment register 10 is used for the application global memory (**`appl_ctl_heap_sz`**) if the **`INTRA_PARALLEL`** or connection concentrator or DPF is enabled.

  – Segment register 11 for application/agent communication heap (**`aslheapsz`**).
- Segment register 12 is used to attach to the shared segment used for DB2 trace.

Important: Problems typically occur when there is a conflict between segments needed for different purposes, since there are only a finite number of segments that can be attached and they must all fit within the configuration.

For example, if DB2 shared memory allocated is so large (7 x 256 MB) as to attach segments 4 through 10, and **INTRA_PARALLEL** is turned on, then a connect will fail with a SQL0987 return code because the application control heap (**appl_ctl_heap_sz**) must be attached at segment 10, which is already taken by the DB2 shared memory.

Therefore, while Figure 5-2 on page 341 implies that up to 7 segment registers are available for DB2 shared memory, configuration or use of other features such as **INTRA_PARALLEL** and others can reduce this number to 5.

By understanding which segments will be required based on the configuration, you should be able to understand and resolve most or all of these problems.

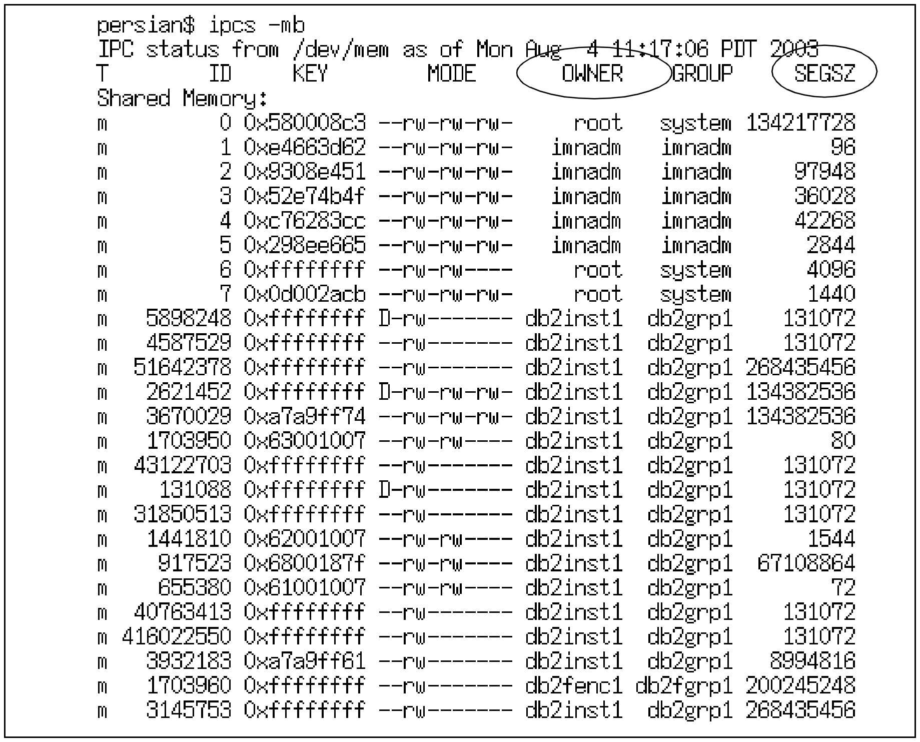
Refer to the IDUG 2001 presentation "Where Did My Memory Go? - Memory Usage in UDB" by Ian Maione for further details.

Extended Storage (database configuration parameters **estore_seg_sz** and **num_estore_segs**) is a DB2 feature whereby a large (typically >4 GB) area of shared memory can be used as an extended buffer pool. When extended storage is enabled, a number of shared memory segments will be created at the AIX level, but only one of these will be attached by a DB2 agent at a particular time. When a buffer pool page needs to be referenced in a different segment, then the current segment is detached and the required segment attached at runtime. There is an adverse performance impact with this approach, since page table entries need to be flushed and recreated when the segment registers are changed.

Tip: Use 64-bit DB2 to access large amounts of physical memory for heaps.

When DB2 processes start, they allocate shared memory segments. If they are already allocated, processes attach to the existing segments. The owners and sizes (SEGSZ in bytes) of shared memory segments can be displayed using the following command, as shown in Figure 5-3 on page 344:

```
ipcs -mb
```



```
persian$ ipcs -mb
IPC status from /dev/mem as of Mon Aug  4 11:17:06 PDT 2003
T        ID     KEY        MODE        OWNER    GROUP    SEGSZ
Shared Memory:
m         0 0x580008c3 --rw-rw-rw-     root   system 134217728
m         1 0xe4663d62 --rw-rw-rw-   imnadm   imnadm        96
m         2 0x9308e451 --rw-rw-rw-   imnadm   imnadm     97948
m         3 0x52e74b4f --rw-rw-rw-   imnadm   imnadm     36028
m         4 0xc76283cc --rw-rw-rw-   imnadm   imnadm     42268
m         5 0x298ee665 --rw-rw-rw-   imnadm   imnadm      2844
m         6 0xffffffff --rw-rw----     root   system      4096
m         7 0x0d002acb --rw-rw-rw-     root   system      1440
m   5898248 0xffffffff D-rw------- db2inst1  db2grp1    131072
m   4587529 0xffffffff --rw------- db2inst1  db2grp1    131072
m  51642378 0xffffffff --rw------- db2inst1  db2grp1 268435456
m   2621452 0xffffffff D-rw-rw-rw- db2inst1  db2grp1 134382536
m   3670029 0xa7a9ff74 --rw-rw-rw- db2inst1  db2grp1 134382536
m   1703950 0x63001007 --rw-rw---- db2inst1  db2grp1        80
m  43122703 0xffffffff --rw------- db2inst1  db2grp1    131072
m    131088 0xffffffff D-rw------- db2inst1  db2grp1    131072
m  31850513 0xffffffff --rw------- db2inst1  db2grp1    131072
m   1441810 0x62001007 --rw-rw---- db2inst1  db2grp1      1544
m    917523 0x6800187f --rw-rw---- db2inst1  db2grp1  67108864
m    655380 0x61001007 --rw-rw---- db2inst1  db2grp1        72
m  40763413 0xffffffff --rw------- db2inst1  db2grp1    131072
m 416022550 0xffffffff --rw------- db2inst1  db2grp1    131072
m   3932183 0xa7a9ff61 --rw------- db2inst1  db2grp1   8994816
m   1703960 0xffffffff --rw------- db2fenc1 db2fgrp1 200245248
m   3145753 0xffffffff --rw------- db2inst1  db2grp1 268435456
```

Figure 5-3 Output of ipcs -mb command

When DB2 shuts down, it normally releases these segments and they disappear. If a problem occurs while shutting down, segments can be left around, and these can be identified since the OWNER of these segments corresponds to the DB2 instance owner. A restart of DB2 normally fails in such cases; errors are recorded in <instance>/sqllib/logs/db2start.<timestamp>.errlog and <instance>/sqllib/db2dump/db2diag.log.

These segments may removed by using the **`ipcrm`** command after ensuring, via the **`ps -ef | grep db2`** command, that there are no DB2 processes running.

DB2 uses other shared resources, such as semaphores and message queues. These can also be viewed using the **`ipcs`** command with **`-s`** or **`-q`** options. They should also normally be released when DB2 shuts down.

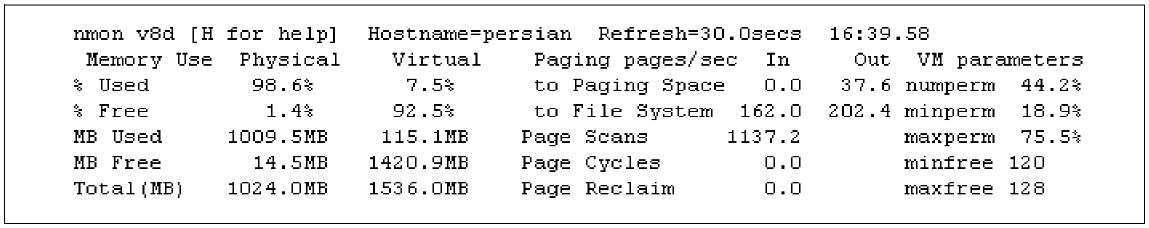
DB2 also provides the **`ipclean`** command. This command has a number of options for cleaning up shared memory, semaphores, and message queues. You can view the command options by using the following command:

```
ipclean -?
```

General recommendations

We recommend the following general best practices:

1. Use **vmtune** to balance process and file memory using **minperm** and **maxperm**; the defaults are 20% and 80%, respectively. Monitor paging rates using a tool such as **nmon**, as shown in Figure 5-4.



```
nmon v8d [H for help]  Hostname=persian  Refresh=30.0secs  16:39.58
 Memory Use  Physical    Virtual    Paging pages/sec  In     Out  VM parameters
% Used        98.6%       7.5%      to Paging Space   0.0   37.6 numperm  44.2%
% Free         1.4%      92.5%      to File System  162.0  202.4 minperm  18.9%
MB Used     1009.5MB   115.1MB    Page Scans      1137.2        maxperm  75.5%
MB Free       14.5MB  1420.9MB    Page Cycles        0.0        minfree 120
Total(MB)   1024.0MB  1536.0MB    Page Reclaim       0.0        maxfree 128
```

Figure 5-4 nmon memory statistics

   – If your system mostly uses raw disk containers, try setting **minperm** and **maxperm** to 10% and 20%, respectively, as follows:

     `/usr/samples/kernel/vmtune -p 10 -P 20`

   – If your system mostly uses system-managed or database-managed file containers, try setting **minperm** and **maxperm** to 15% and 50%, respectively, and monitor page fault rates as described here:

     - If process paging rates are less than 5 to 10 page ins or page outs per second for extended periods, reduce the values of **minperm** and **maxperm** and repeat the monitoring process.
     - If the filesystem paging rates are "high" (this depends on the number of filesystems), then increase the values of **minperm** and **maxperm** and repeat the monitoring process.

DB2-specific recommendations

We recommend the following DB2-specific best practices:

1. If your system never or rarely pages and you have lots of memory and paging space, set **DB2MEMDISCLAIM=NO**. Otherwise, set **DB2MEMDISCLAIM=YES** and set **DB2MEMMAXFREE=8000000**. DB2 registry variables are set via the **db2set** command.
2. Setting the DB2 registry variable **DB2_PINNED_BP = YES** (default is **NO**) causes all database global memory to be pinned; it will not be paged out. This provides more consistent response time performance, but can impact other processes that will bear the brunt of paging instead. It may also adversely affect filesystem performance.

Since most DB2 servers have a large amount of main memory that cannot be fully used by DB2, consider setting **DB2_PINNED_BP = YES** to take advantage of this unused memory and to improve the overall performance of your database environment.

3. Similarly, if the system uses SMS or DMS file containers, set **DB2_MMAP_READ=OFF** and **DB2_MMAP_WRITE=OFF** (the default is **ON** for both variables) so as to not use memory mapped files. This frees up one segment register and takes advantage of the filesystem buffers.

Note: Memory mapped files are a UNIX technique whereby programs can “map” files or parts of files into memory and then address those parts as though they were simply program data instead of having to read and write the data file each time it is accessed. If a system has plenty of memory, this offers some performance and programming advantages.

Refer to *IBM DB2 Universal Database Administration Guide: Performance V8,* Appendix A, “Registry and Environment Variables” for default values and valid settings.

5.2.3 Disk and filesystem considerations

In this section, we provide the following:

1. Disk subsystem review
2. Disk subsystem recommendations
3. Filesystem recommendations
4. DB2-specific recommendations

Disk subsystem review

Disk subsystems tend to have the most significant performance impact on applications by virtue of their electromechanical nature; in the time it takes to move heads across disks, CPUs can execute millions of instructions.

A good disk subsystem design not only involves selecting the right number, size and performance characteristics for the disks, but also the topology used in connecting these disks via disk adapters and adapter buses.

In a balanced system, each bus such as a SCSI or Fibre bus will support the bandwidth and I/Os per second capabilities of the disks on that bus. In addition, the PCI bus that the disk adapters are connected to should be able to support the bandwidth requirements of those adapters and whatever else is in the PCI slots and so on. Figure 5-5 on page 347 illustrates a simplified view of a disk subsystem architecture and potential bottleneck areas.

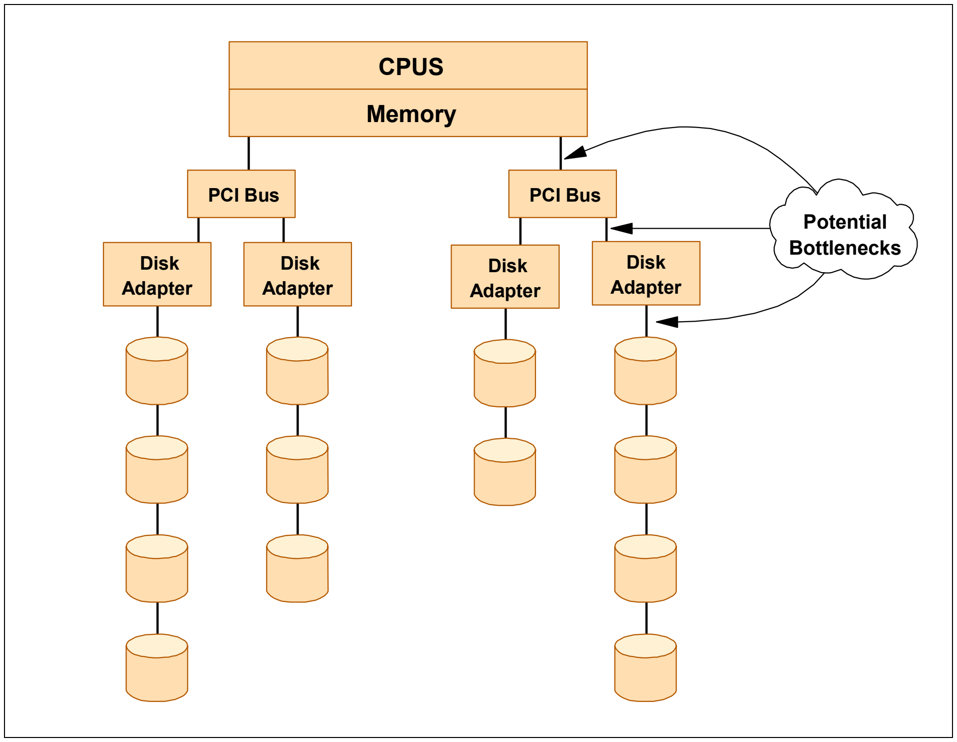


Figure 5-5 Simplified view of disk subsystem architecture

In most cases, disk subsystem design is done with little or no DBA involvement. However, it is important for DBAs to understand potential areas of bottlenecks in the disk subsystem in order to be able to identify and resolve performance problems relating to the disk subsystem.

The economics of disk technology are driving configurations towards fewer and larger disks. However, there are performance benefits in using more, smaller disks. These equations must be solved for each particular system.

Refer to *IBM DB2 UDB DB2 V8 Administration Guide: Performance*, SC09-4821, Chapter 2 for details on how DB2 lays out control information and database objects such as tables across disks when creating a database.

Note: Chapter 9 of the IBM Redbook *Database Performance Tuning on AIX*, SG24-5511, provides an extensive discussion of the design of disk subsystems for database environments.

The next subsections provide a brief overview of the following:

- Disk technology fundamentals
- RAID
- SAN storage
- AIX Logical Volume Manager
- Mirroring and striping data
- Striping data and prefetch
- 64-bit filesystem performance

Disk technology fundamentals

Disk performance is characterized by speed (revolutions per minute(rpm)), head access times (milliseconds), and data transfer rates (MB or GB per second).

- Current disks support high speed rpm (7200, 10000, and 15000 rpms), which are significantly higher than first generation 3600 rpm disks. Higher speeds give better performance because the linear speed of the head over the track results in higher transfer rates.
- Head access time refers to the time it takes to move the disk's read/write head from one track to another. The conventional measurement (seek time) is the number of milliseconds needed to move the head across half the tracks on a disk platter. The smaller the head access time, the better performance is delivered.

The length of a disk track[3] increases from the center of a disk platter to the outer edge. Early technology disks had the same number of sectors on the inner tracks and outer tracks, thus limiting the amount of information recorded to the capacity of the innermost track; this resulted in underutilization of the capacity of the outer tracks. Modern disks use a concept called "zone recording" where inner zones have fewer sectors per track, and outer zones have a larger number, thereby significantly increasing the amount of data that can be stored on a disk. There are typically three or more zones on a disk.

A consequence of zone recording is that you can transfer more data to or from an outer zone track without moving the disk heads, compared to an inner zone track. For the highest possible performance, place performance critical data on the outer edges of the disk, and use the rest of the disk for infrequently accessed data.

However, when you need to place a number of files or filesystems on a single spindle, place the most frequently accessed one in the central zones. This gives the highest probability that the disk heads will be over, or moving towards, your frequently accessed data.

[3] This, effectively, is the circumference of a circle.

Note: The placement rules do *not* apply if you are using virtual or logical disks, for example, in a SAN.

RAID

Redundant Array of Independent Disk, or RAID, technology implements virtual disks from a number of physical disks. RAID combines multiple physical disks into a single virtual disk that provides data protection and increased performance. RAID can be implemented in software such as in Logical Volume Manager, or in hardware such as various disk adapters, IBM ESS, and FAStT storage systems.

We recommend that where RAID is provided by hardware, it should normally be used in preference to the software solution (such as Logical Volume Manager).

Table 5-1 on page 351 provides a brief overview of the various types of RAID and their characteristics.

RAID 0

RAID 0, or striping, does not provide data redundancy in a disk subsystem; the primary goal of RAID 0 is to improve I/O performance. When data is written on a striped logical disk (or LUN), the data is divided into small pieces known as *chunks* or *stripe units*. These units are then written to the disks in parallel, in a round-robin fashion. During a read operation, the same units are read back in parallel.

Performance is related to the stripe size and number of disks in the array. Typically, performance will be better with larger numbers of disks and in multi-user environments.

The main problem with RAID 0 is obviously that any disk failure in the array will render the data on the remaining disks unusable. For that reason, only consider RAID 0 if you require good performance but are not concerned about availability.

RAID 1

RAID 1, or mirroring, simply means keeping a copy of the data on one or more additional disks to protect against data loss in the event of a single disk failure. To improve availability, the copies should be kept on separate physical disks, which ideally should be attached to separate I/O adapters.

To maximize availability, the disks should be in separate disk cabinets to protect against events such as a power loss to one of the disk cabinets.

In the event of a disk failure, read and write operations will be directed to the mirrored copy of the data. In performance terms, mirroring has some advantages over striping. If a read request is received and the primary data copy is busy, then the read is directed to the mirrored copy. However, write performance is penalized because each copy requires a separate write.

Notes:

- AIX mirroring provides several scheduling policies for mirroring. These have different performance and data integrity characteristics. Consult the LVM documentation for additional information.
- In AIX, when using LVM to implement mirroring data, *only* the logical volumes are mirrored, not the physical disk. Mirroring is not a tool to use to provide a disk copy.

RAID 5

In a RAID 5 array, data and parity are spread across all disks. For example, in a 5+P RAID 5 array, six disks are used for both parity and data. In this example, five-sixths of the available space is used for data and one-sixth is used for parity.

Because of the parity data used by RAID 5, each write to the array will result in four I/O operations; this is known as the RAID 5 write penalty:

1. Read old data
2. Read old parity
3. Write new data
4. Write new parity

If the database activity is very write-intensive, then the write penalty will become a factor. However, hardware caches, which exist on ESS and FAStT disk subsystems as well as some RAID adapters, can reduce the effects of this penalty. As well as simply caching I/O requests, it will frequently be possible to calculate new parity because data and parity blocks remain in the cache, therefore avoiding the need to read from the disks.

RAID 5 is commonly chosen because it provides an extremely good price/performance ratio combined with good availability.

RAID 10

Sometimes called RAID 0+1, this RAID level provides better data availability at the cost of extra disks. Consider a striped logical volume, where failure of one disk renders the entire logical volume unusable. RAID 10 provides the ability to mirror the striped logical volumes to prevent this. When data is written to the disks, the first data stripe is data and the subsequent stripe copies (maximum of three copies, including first copy) are the mirrors and are written to different physical volumes.

Table 5-1 A summary of RAID types

RAID type	Method and benefit
RAID 0	► Data is written in stripes over 'N' disks. ► No redundancy or data parity. ► Requires one physical I/O per logical I/O (or more, depending on transfer size and stripe size. ► Provides performance improvement for multi-user access.
RAID 1	► Same data written to two or more ("M") disks for data protection (more often known as *mirroring*). ► Requires "M" physical I/Os.
RAID 5	► Data written in stripes over "N" disks with parity written to $(N+1)^{th}$ stripe. ► Parity stripe moves sequentially from disk to disk. ► Requires 4 physical I/Os.
RAID 10	► Effectively RAID 0 + RAID 1; that is, provides data protection and performance. ► Assuming 2-way mirror, requires 2 phys. I/Os.

Recommendations about using RAID in conjunction with DB2 are discussed in "DB2-specific recommendations" on page 345.

SAN storage

Storage Area Networks, or SAN, are increasingly being used to store large databases. AIX eServer pSeries systems support a variety of IBM and OEM devices. They are normally connected by multiple fiber optic cables; however, the high throughput of fiber (2 Gigabits per second) does not always translate into disk throughput.

Tip: The Fibre Channel Industry Association preferred spelling is "fibre channel" when referring to standards or fibre channel products, and "fiber" when referring to fibre optics in general.

Potential issues include:

- Some devices and device drivers do not support multi-path I/O (load balancing of I/Os) across multiple disk adapters. Therefore, all I/O to the logical disk is via a single fiber despite multiple physical connections.

- Although the fiber can support 2 Gb per second, the SAN device may be limited in the number of individual I/Os per second it can support. This may be a significant concern when the I/Os are small.

Note: For optimal performance, the DBA needs to influence how logical disks (LUNS) are ultimately mapped to physical disks in the SAN. The trend, however, is towards a quality of service (QoS)-driven approach to logical volume allocation, and the DBA's ability to directly control device configurations is diminishing.

Recommendations about using SAN in conjunction with DB2 are discussed in "DB2-specific recommendations" on page 345.

A number of tools are available to monitor disk subsystem performance:

- Consider using `iostat` and `nmon` for an instantaneous view of the LUNs, not the physical disks. These tools may also be used to record data by redirecting `iostat` to a file. `sar` and `filemon` can also record data in a file. "Monitoring and problem determination tools" on page 363 describes some of these tools.
- Tools provided by the SAN storage subsystem provide a performance perspective from the storage device end that can be compared with operating system performance data.

Enterprise Storage Server (ESS)

The IBM Enterprise Storage Server® (ESS) is a SAN-connected disk storage subsystem capable of holding terabytes of data. It can support multiple connections, each with multipath I/O in conjunction with AIX (and Windows) operating systems. AIX provides concurrent access to an ESS logical disk via each path of a multipath connection.

The ESS has a large cache that logically sits between the operating system and ESS disks. This cache buffers both disk reads and writes.

Disks in the ESS are organized in "ranks" of, typically, seven disks forming a "6+P" RAID5 array. The latest ESS model also implements a RAID10 array which may offer superior performance in heavy update-oriented environments. Ranks are split up into "logical units" or LUNs. LUNs correspond to logical disks at the operating system level.

It is possible to treat the ESS as a black box that simply provides a number of fixed sized logical disks for use by DB2. While this approach may also provide simple administration, for optimal performance it is necessary to understand how LUNs are mapped to ranks and the access paths to those ranks.

The StorWatch Specialist for ESS provides a graphical PC-based tool to view and administer the ESS. This is normally only available to the ESS administrator.

ESS and SMS considerations

LUNs can be used as SMS or DMS containers. For an SMS or DMS FILE container, you will need a logical volume (to contain the filesystem) created on the LUN by LVM (see “AIX Logical Volume Manager” on page 355). Do not create LVM volumes as RAID, because this would place RAID on top of RAID, which will have poor performance characteristics.

To minimize filesystem lock contention on update-oriented SMS containers, create several SMS containers per rank. For an explanation of lock contention, see “64-bit filesystem performance” on page 357. However, if you are using tablespace striping, to maximize performance, ensure that the tablespace’s containers are not in the same rank. This applies for both SMS and DMS containers.

ESS and DMS considerations

If you use AIX LVM to create logical volumes for DMS FILE or DEVICE containers, the Logical Volume Manager will allow you to create the volume across multiple hdisks (or LUNs, in the ESS case). While you would normally do this in a non-RAID environment, it is not appropriate in this case because the underlying LUNs are already RAIDed and there is no performance advantage over using only one LUN per volume.

Take advantage of the inherent striping capabilities of DB2 by placing containers on separate LUNs on separate ranks wherever possible. This will balance the I/O load.

If you use DMS DEVICE containers, you do not need to use LVM to create logical volumes for them. You can use raw hdisk or vpath objects directly. For example, the corresponding container statement for a multipath device would look like the following:

```
DEVICE ‘/dev/rvpath17’ 102400
```

ESS and DB2 configuration

DB2 stripes across containers at the tablespace level, and the stripe width is the extent size. ESS LUNs are striped with a stripe width of 32 KB. Selecting container extent sizes that are multiples of 32 KB will ensure multiple disks in each LUN are accessed when a prefetch occurs.

A reasonable starting point is to choose an extent size equal to the number of physical disks in the LUN; for example, for a 6+P rank, set extent size = 6*32 KB or 196 KB. Alternatively, you may set extent size to 32 KB and prefetch to 196 KB—this is better for OLTP environments.

Attention: Large extents can result in inefficient space utilization. For example, MDC dimension columns use one extent for each unique value. Consequently, a poor choice of dimensions can consume large amounts of storage.

FAStT disk subsystems

The IBM FAStT disk subsystem is also a SAN-connected disk storage subsystem, but with significant differences compared to the ESS.

There are several models of FAStT subsystem. The common features include internal dual redundant loops of disks and two or more fiber connections to a host system. FAStT provides RAID 0, 1, 3, 5 and 10, and LUNs can be converted between RAID types transparently to the host given sufficient spare disk capacity in the FAStT. Striped RAID uses a default stripe width of 64 KBytes, but values from 8 KBytes to 256 KBytes can be used. There is a limited ability to change the stripe size transparently to the host.

The base model, FAStT 200, uses a pair of 1 GBit/second fiber connections, while other models use 2 GBit/second connections. FAStT subsystems support dual paths from AIX; however, I/O is not concurrent. Rather, a round robin policy provides load balancing across the two fibers. Therefore, I/O rates and bandwidth may be significantly lower than an ESS. This will depend on the fiber speed, patterns of I/Os, and number of disks in the subsystem.

The subsystem presents logical units to the operating system in a similar manner to the ESS.

There is no concept of “ranks” in a FAStT. Internally, all disks are dual attached to a pair of loops and can be accessed from either loop. RAIDed LUNs are normally created by constructing them from N disks, with half accessed via loop A and the remainder accessed via loop B.

A graphical PC-based tool called FAStT Storage Manager is used to view and manage the FAStT subsystem.

FAStT and SMS containers

The factors mentioned in “ESS and SMS considerations” on page 353 also apply to FAStT. Specifically, to minimize filesystem lock contention, use multiple filesystems for SMS containers if possible.

FAStT and DMS containers

There appears to be little benefit in creating multiple DMS device containers per tablespace within a FAStT subsystem, as the containers will share a common access path to and within the FAStT. However, some FAStT models are configurable with a pair of dual attached disk loops. In this case, a container on each loop may provide improved performance.

FAStT and DB2 configuration

The same factors mentioned in “ESS and DB2 configuration” on page 353 apply.

AIX Logical Volume Manager

AIX includes a sophisticated Logical Volume Manager (LVM) which implements logical volumes on physical disks. These logical volumes can be used for filesystems or as raw devices. The LVM can also mirror or stripe logical volumes using software.

Note: Many storage systems (SAN or SSA) also implement mirroring and/or striping. This may be in the device adapter or in the storage subsystem, such as IBM ESS.

Refer to the following Redbooks for detailed information about LVM:

- *AIX Logical Volume Manager, From A to Z: Introduction and Concepts*, SG24-5432
- *AIX Logical Volume Manager from A to Z: Troubleshooting and Commands*, SG24-5433

Mirroring and striping data

Mirroring protects against data loss due to simple disk media or access errors, while disk striping is typically used to increase disk bandwidth or throughput beyond that of a single disk.

Mirroring and/or striping may be implemented by DB2, by the LVM, or by the disk storage subsystem. The choice of a particular approach depends upon the performance and data integrity requirements of the database, and on familiarity and ease of management, as follows:

- When mirroring is implemented by the LVM, the operating system must perform two or more[4] physical I/Os for each logical write I/O.
- A DB2 tablespace associated with multiple containers implements striping by default in DB2 V8. DB2 implements striping at the container level, and the stripe width is the extent size. You can add containers to a tablespace to

[4] This depends upon the number of mirrors.

increase the size of the tablespace. The stripe width remains constant, but the number of strips increases.

Alternatively, the LVM may be used to provide a striped logical volume. LVM stripes data within a single logical volume constructed from a number of physical volumes. The stripe width is specified when the volume is created and cannot be altered. Nor can you add physical volumes to the logical volume.

Note: It is permissible to combine DB2 and LVM striping for a given tablespace, in which case the tablespace extent size must be an integer multiple of the logical volume stripe size, and all volumes must use the same stripe size.

Table 5-2 describes some of the considerations in choosing between the technologies available.

Table 5-2 Mirroring/Striping decision matrix

Function or subsystem	Mirror?	Stripe?
Disk adapter or SAN	► Typically implemented by adapter or SAN device. ► Use to protect data in place of other technology.	► Typically implemented by adapter or SAN device. ► Can provide data protection in place of mirroring. If so, use instead of mirroring.
Logical Volume Manager	► Use to protect data if no adapter/SAN mirroring available.	► Can be beneficial for DB2 performance. ► Do not use if adapter/SAN implements striping instead.
DB2	► Mirroring log files may improve performance.	► Use multiple containers per tablespace for maximum performance.

Striping data and prefetch issues

Both DB2 and AIX filesystems will prefetch data if they detect sequential I/O. It is critical for performance that DB2 prefetch processes interact efficiently with any data striping regardless of whether it is implemented using containers, LVM, or SAN.

As mentioned earlier, DB2 automatically implements striping at the tablespace level when a tablespace is created with multiple containers, or when an existing tablespace is altered to add containers[5].

Note: The operating system will not prefetch data if containers are raw devices or raw logical volumes. However, DB2 can prefetch from raw devices or volumes.

If the containers are SMS or DMS file containers, the operating system will prefetch data if it detects sequential reads. Prefetch is driven by two AIX parameters: **`minpgahead`** (default 2 pages) and **`maxpgahead`** (default 8 pages). The operating system will read ahead **`minpgahead`** pages and, on detecting continued sequential access, will double the number of pages read ahead on each read until it reachs **`maxpgahead`**.

Note: These AIX parameters are common to all filesystems, and its impact on non-DB2 filesystems must be evaluated.

Recommendations about using striping and prefetch in conjunction with DB2 are discussed in "DB2-specific recommendations" on page 345.

64-bit filesystem performance

There are some filesystem performance benefits that flow from using 64-bit AIX, since AIX5.1 and later support a new filesystem type, JFS2, exclusively on the 64-bit operating system.

SMS and DMS file containers on JFS2 filesystems offer performance benefits over the more common JFS filesystems.

Enhancements implemented in AIX5.2 ML1 provide further performance benefits for SMS and DMS containers defined as files. A feature called "Concurrent I/O" allows multiple processes such as DB2 page cleaners to write to the same file (container) without holding an exclusive lock on the file. This typically increases database performance to the level of raw devices, while maintaining the manageability of files and filesystems. Concurrent I/O can be implemented simply by mounting the filesystems containing containers with the **`-o cio`** option. When it is not practical to mount the whole filesystem in this mode, a subdirectory of the mount point can be over-mounted with these options.

[5] Altering a tablespace to add containers may invoke a background rebalancing process which adversely affects performance. Use the BEGIN NEW STRIPE SET option in the ALTER TABLESPACE statement to allocate a new container above the high water mark, such that a rebalance will not occur.

Attention: This feature is expected to become available in DB2 V8.1 FP4.

General file system performance

When the operating system needs to read a logical volume, it allocates a buffer called a pbuf. If none are available, the read is delayed until one is available. A similar situation arises when reading from a filesystem; these buffers are called fsbufs. Default values of pbufs and fsbufs are normally adequate; however, AIX 5.2 provides a tool, **`vmstat`**, to monitor these buffers. Do not over-configure pbufs or fsbufs, because they are pinned in memory and cannot be paged.

Disk subsystem recommendations

We recommend the following disk subsystem best practices:

1. Try to spread the I/O load across as many disks and adapters as possible. Balance this load as much as possible across disks and adapters.
2. Place data that has high bandwidth requirements or high I/O requirements on a separate disk and, if possible, a separate I/O bus.
3. Monitor disk subsystem performance with **`iostat`** or **`nmon`**.
4. Place files or data with high access performance requirements on the outer edges of disks. If this is not practical, place them in the middle between the inner and outer tracks.
5. Monitor disk utilization using **`iostat`** to make sure it is normally less than 75% per physical disk. If it exceeds 75%, use **`filemon`** to identify the busy volumes or files. If disk utilization is high but individual volumes on the disk are not busy, consider the possibility that two logical volumes on the disk may be “far apart” and the disk is busy because the heads are frequently moving backwards and forwards between the volumes. While some authors recommend a ceiling of 40% rather than 75% disk utilization, the important factor is to make sure utilization is as balanced as possible across all disks.
6. To protect your data, use SAN or LVM mirroring techniques. Assuming the SAN storage implements some sort of RAID, use the SAN if you have one; otherwise, use LVM volume mirroring.
7. If you have a large number of DMS containers on raw LVM logical volumes, consider increasing **`lvm_bufcnt`**, a variable that defines the number of buffers available for raw I/Os. The buffers size is 128 KB and the default value is 9.

   You will typically only need to increase this value if you are doing very large I/Os and have many very fast disks. There is no direct way to monitor this variable.

   If running AIX 5.2, use the **`ioo`** command; otherwise, use **`vmtune`**.

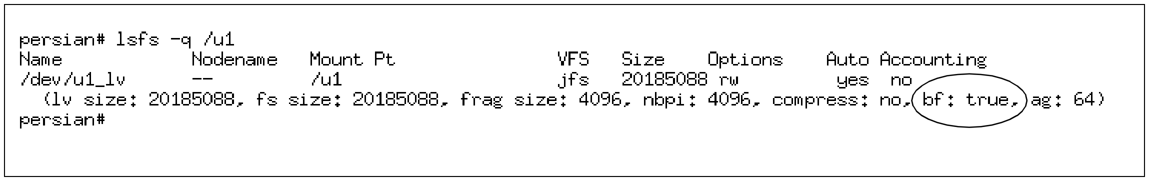
8. If you are using a SAN, determine the mapping of logical to physical disks. Verify that the different disks are actually physically separate, on separate buses within the SAN device, and so on. If you still have disk performance problems, record disk performance data using one of the available tools and coordinate these measurements with the SAN administrator's measurements for the same period. Comparing these should highlight any unexpected configuration problems.

Filesystem recommendations

We recommend the following filesystem best practices:

1. The default filesystem type limits files to a maximum of approximately 2 GB. We recommend creating filesystems as "large file enabled", because this will permit individual containers to exceed 2 GB.

   Note: You cannot convert an ordinary filesystem to large file enabled. However, you can verify whether a filesystem is large file enabled by using the **`lsfs`** command, as shown in Figure 5-6.

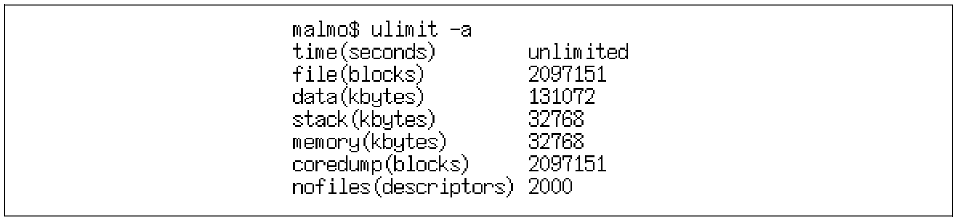


```
persian# lsfs -q /u1
Name            Nodename   Mount Pt               VFS   Size    Options    Auto Accounting
/dev/u1_lv      --         /u1                    jfs   20185088 rw         yes  no
  (lv size: 20185088, fs size: 20185088, frag size: 4096, nbpi: 4096, compress: no, bf: true, ag: 64)
persian#
```

Figure 5-6 lsfs command output

   A large file enabled filesystem is shown by `bf: true` in the output.

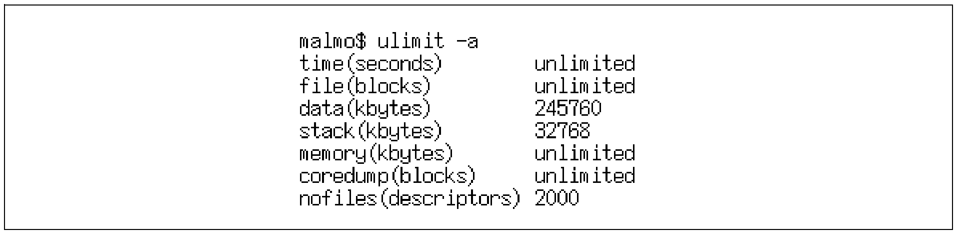
2. The system default **`ulimit`** settings prevent users creating files larger than 2 GB. Check **`ulimit`** settings with the command **`ulimit -a`** as shown in Figure 5-7.



```
malmo$ ulimit -a
time(seconds)        unlimited
file(blocks)         2097151
data(kbytes)         131072
stack(kbytes)        32768
memory(kbytes)       32768
coredump(blocks)     2097151
nofiles(descriptors) 2000
```

Figure 5-7 Default ulimit settings

The value needs to be altered for all DB2 instance, admin and fenced user logins. The value should be set to -1, that is, allowing DB2 to create files of unlimited size as shown in Figure 5-8.



```
malmo$ ulimit -a
time(seconds)        unlimited
file(blocks)         unlimited
data(kbytes)         245760
stack(kbytes)        32768
memory(kbytes)       unlimited
coredump(blocks)     unlimited
nofiles(descriptors) 2000
```

Figure 5-8 ulimit values for db2 processes

3. If you are using SMS or DMS file containers, ensure that JFS logs are not on the same disks as the corresponding filesystems. Each time a filesystem is modified, an entry is written to the JFS log. If the filesystem and JFS log are on separate disks, this will avoid disk head movement.
4. Monitor the system to ensure that there are enough operating system buffers for disk I/Os and that the operating system is not stalling, waiting for buffers to become free. The monitoring method used depends on the operating system version installed, but in each case, ensure that the count of “blocked” buffers (each of which corresponds to a brief stall by the operating system) is not increasing. Table 5-3 lists and describes some of the more relevant operating system variables.

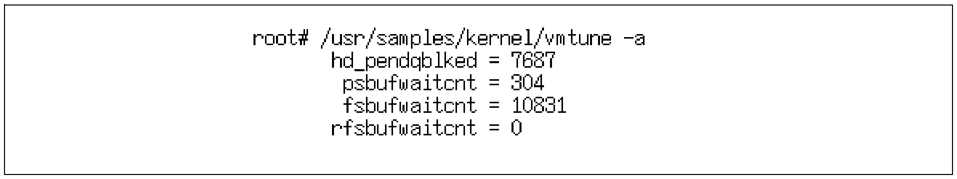
Table 5-3 Operating system I/O variables

Variable	Description
hd_pendkblked	A count of the number of times the operating system was not able to acquire a buffer for physical disk I/O.
hd_pbuf_cnt	A variable defining the maximum number of operating system buffers available for disk I/O at any one time.
fsbufwaitr_cnt	A counter of the number of times the operating system was not able to acquire a buffer for filesystem I/O.
numfsbufs	A variable defining the maximum number of operating system buffers available for filesystem I/O at any one time.

Use the following commands:

– With AIX 4.3.3 or AIX 5.1, run the command **`vmtune -a`** as shown in Figure 5-9 from time to time and check that the variable **`hd_pendkblked`** is not increasing. If it is, then **`hd_pbuf_cnt`** needs to be increased.

  Also check that the variable **`fsbufwait_cnt`** is not increasing. If it is, then **`numfsbufs`** needs to be increased.

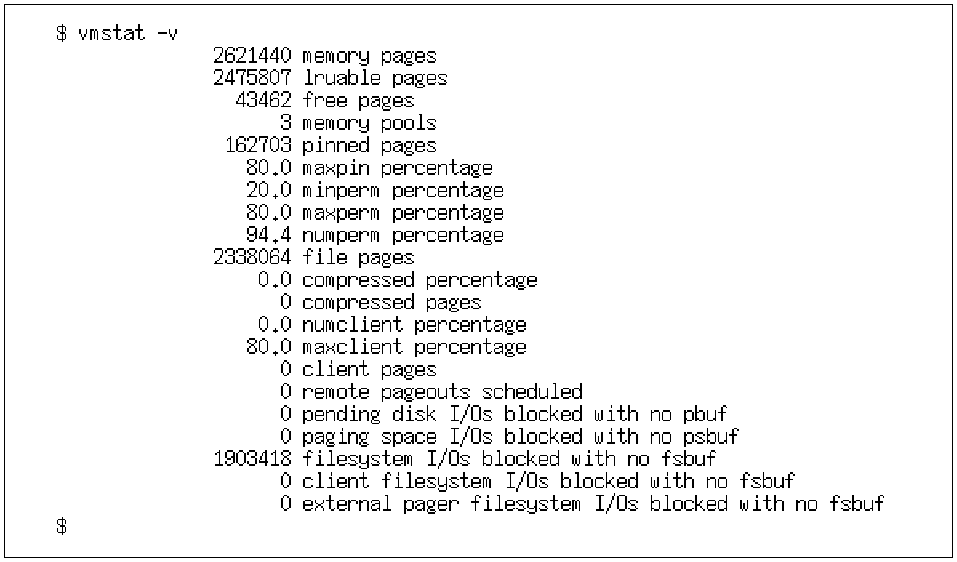


```
root# /usr/samples/kernel/vmtune -a
       hd_pendqblked = 7687
        psbufwaitcnt = 304
        fsbufwaitcnt = 10831
       rfsbufwaitcnt = 0
```

Figure 5-9 vmtune -a output (AIX 4.3.3)

– With AIX 5.2, run the command **`vmstat -v`** as shown in Figure 5-10 from time to time and check that the count of “pending disk I/Os blocked with no pbuf” is not increasing> If it is, then **`hd_pbuf_cnt`** needs to be increased.

  Also check that “filesystem I/Os blocked with no fsbuf” is not increasing. If it is, then **`numfsbufs`** needs to be increased.



```
$ vmstat -v
             2621440 memory pages
             2475807 lruable pages
               43462 free pages
                   3 memory pools
              162703 pinned pages
                80.0 maxpin percentage
                20.0 minperm percentage
                80.0 maxperm percentage
                94.4 numperm percentage
             2338064 file pages
                 0.0 compressed percentage
                   0 compressed pages
                 0.0 numclient percentage
                80.0 maxclient percentage
                   0 client pages
                   0 remote pageouts scheduled
                   0 pending disk I/Os blocked with no pbuf
                   0 paging space I/Os blocked with no psbuf
             1903418 filesystem I/Os blocked with no fsbuf
                   0 client filesystem I/Os blocked with no fsbuf
                   0 external pager filesystem I/Os blocked with no fsbuf
$
```

Figure 5-10 vmstat -v output (AIX 5.2)

Note: The system needs to be rebooted for the change to **`hd_pbuf_cnt`** to take effect, while the filesystem will need to be unmounted and remounted before the change to **`numfsbufs`** takes effect.

DB2-specific recommendations

We recommend the following DB2-specific best practices:

1. Place the catalog tablespace (SYSCATSPACE) and the DB2 logs on separate disks. After creating a database, create new system temporary tablespaces on separate disks and drop the default system temporary tablespaces.
2. Tablespaces constructed from multiple containers that are striped logical volumes or striped raw devices (such as a SAN device) should have the DB2 registry variable **`DB2_PARALLEL_IO`** defined for them. The degree of parallelism is determined by the extent size and prefetch size values for the tablespace. The ratio (prefetch size/extent size) should be at least as large as the number of disks in the stripe.
3. With DB2 V8 FP4, consider using filesystems mounted with the “Concurrent I/O” option for SMS and DMS file containers.
4. The **`CREATE TABLESPACE`** statement allows you to specify the **`OVERHEAD`** and **`TRANSFERRATE`** of the disks in your system. This is used to determine the cost of I/O during query optimization.

   The default values 24 and 0.9 do not correspond to current disk technology.

   Attention: The default disk performance numbers in the **`CREATE TABLESPACE`** statement should not be used. Use values obtained from the disk manufacturer of the actual disks.

   For 10 K RPM disks use **`OVERHEAD=8`**, and for 15 K disks use **`OVERHEAD=6`**. **`TRANSFERRATE`** depends on the size of the page. For example, with a 4 KB page on UltraSCSI3 or SAN disks, use **`TRANSFERRATE=0.1`**.

   If the tablespace is spread across a number of disks with different performance characteristics, choose an average of all the values. The UNIX sysadmin should be able to provide disk performance data.
5. If you are using SMS or DMS file containers on striped LVM volumes, consider increasing **`maxpgahead`** to (16 * the number of disks in the volume). This can increase performance by reading multiple disks in parallel. You should also ensure that the **`vmtune`** parameter **`maxfree`** is at least as large as **`maxpgahead`**.

6. If you are using striped DMS tablespaces (multiple containers) on ESS, ensure the extent size is a multiple of the rank's stripe width (normally 32 KBytes).
7. I/O servers are used by database agents for prefetch I/O or asynchronous I/O by utilities (for example, backup or restore). Having more than required does not impose a large load on the system. Configure **`NUM_IOSERVERS`** equal to the number of physical disks used by the database.
8. Asynchronous page cleaners flush dirty pages from the buffer pools to maintain a certain amount of free spaces. The number of page cleaners can have a significant impact on performance. Increase the number of page cleaners (**`NUM_IOCLEANERS`**) if the database is heavily update-oriented or has a large number of buffer pool pages. This is typically set to the number of CPUs on the server.

For additional information about RAID and IBM ESS exploitation with DB2, refer to the IBM Redbooks *Database Performance on AIX*, SG24-5511, and *IBM ESS and IBM DB2 UDB Working Together*, SG24-6262.

5.2.4 Monitoring and problem determination tools

This section provides a brief description of UNIX commands available for monitoring and tuning the performance of the operating system.

Note: In most cases, these commands require root or other privileges only held by the UNIX sysadmin, and *not* by the DBA. Therefore, the DBA will most likely have to coordinate with the UNIX syadmin to monitor the system resources of interest.

Table 5-4 lists tools that are commonly used to monitor various resources of interest (this is not a comprehensive list).

Table 5-4 Suggested tool usage

Performance problem	**Suggested tools**
High CPU utilization	vmstat, sar, nmon,
Not enough memory	vmstat, vmtune, nmon, svmon
Paging and/or swapping	vmstat, vmtune, nmon, lsps, Memory Visualizer
Disk or filesystem	iostat, sar, nmon, filemon

This section covers the following topics:

- Mapping filesystems to physical disks

- filemon
- iostat
- lsps
- nmon
- ps
- System activity recorder (sar)
- svmon
- vmstat
- vmtune

A brief overview of each tool is provided, along with sample output.

Mapping filesystems to physical disks

To determine disk utilization and placement relating to specific DB2 objects, it is necessary to map filesystems to the physical disks on which they reside.

Figure 5-11 shows a sample filesystem[6] associated with a logical volume.

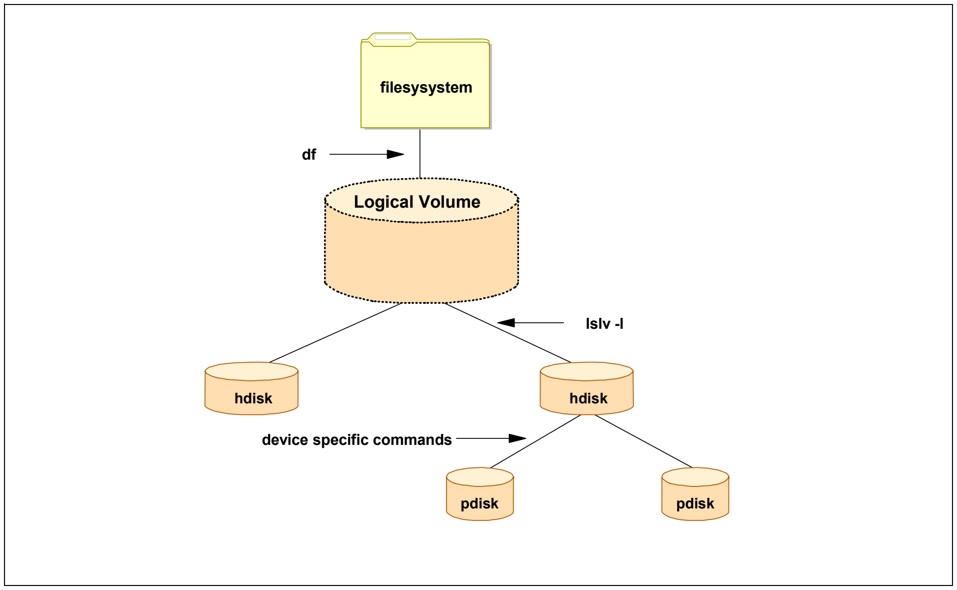


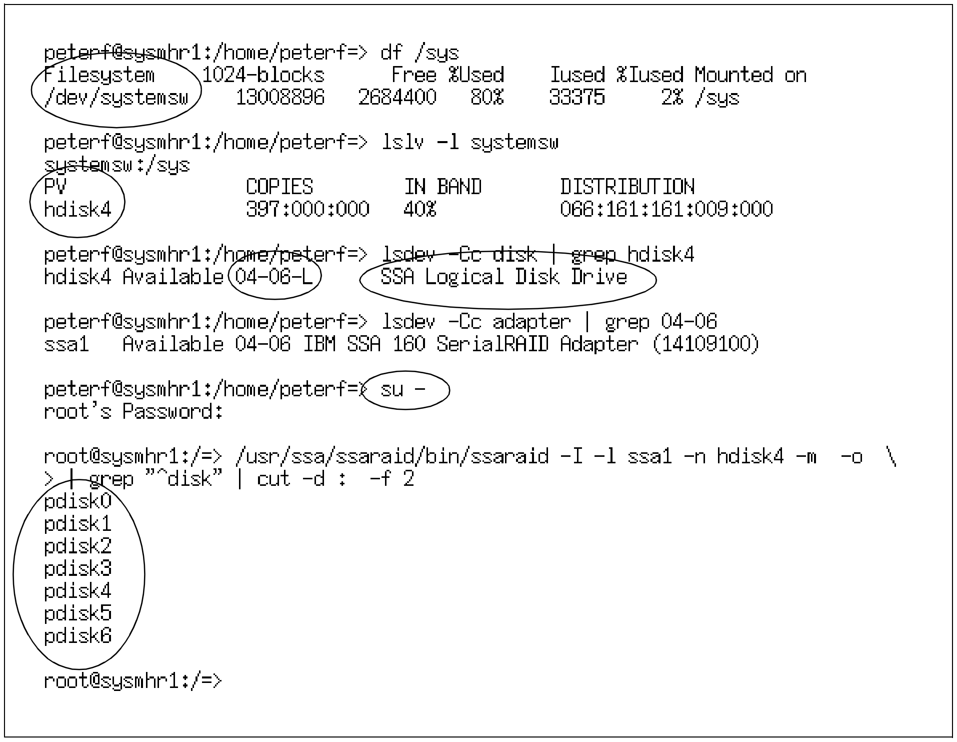
Figure 5-11 Relationship between filesystems and disks

This subsection provides a brief overview of the steps involved in determining the relationship using a number of UNIX commands, as follows:

[6] Adding a filesystem writes essential data structures in the target volume, and it becomes usable after this filesystem is mounted

1. Identify the logical volume associated with a file system by using the **df** command.
2. Identify the “hdisks” associated with the logical volume by using the **lslv -l** command.
3. Identify the “pdisks” associated with a given “hdisk” by using the **ssaraid** command.

For example, in Figure 5-12, we show how to identify the physical disks associated with filesystem /sys.



```
peterf@sysmhr1:/home/peterf=> df /sys
Filesystem    1024-blocks      Free %Used    Iused %Iused Mounted on
/dev/systemsw    13008896   2684400   80%    33375     2% /sys

peterf@sysmhr1:/home/peterf=> lslv -l systemsw
systemsw:/sys
PV                COPIES        IN BAND       DISTRIBUTION
hdisk4            397:000:000   40%           066:161:161:009:000

peterf@sysmhr1:/home/peterf=> lsdev -Cc disk | grep hdisk4
hdisk4 Available 04-06-L     SSA Logical Disk Drive

peterf@sysmhr1:/home/peterf=> lsdev -Cc adapter | grep 04-06
ssa1   Available 04-06 IBM SSA 160 SerialRAID Adapter (14109100)

peterf@sysmhr1:/home/peterf=> su -
root's Password:

root@sysmhr1:/=> /usr/ssa/ssaraid/bin/ssaraid -I -l ssa1 -n hdisk4 -m  -o  \
> | grep "^disk" | cut -d :  -f 2
pdisk0
pdisk1
pdisk2
pdisk3
pdisk4
pdisk5
pdisk6

root@sysmhr1:/=>
```

Figure 5-12 Mapping a filesystem to disks

The **df /sys** command in Figure 5-12 displays the filesystem mounted as /sys. The Filesystem column shows the device on which the filesystem resides. For local filesystems, these are found in /dev. In this case, the device is the logical volume named systemsw.

The **lslv -l systemsw** command[7] in Figure 5-12 lists a single hdisk, named hdisk4, as being associated with the filesystem as reported in the PV column.

[7] The -l option limits the report to the four columns shown.

However, these hdisks are not necessarily real disks, since they may be “virtual disks”; therefore, more discovery is needed.

The **`lsdev -Cc disk`** command in Figure 5-12 on page 365 provides more information about hdisk4. For example, had there been more hdisks associated with the filesystem, then the following steps would need to be executed for each one. The report indicates hdisk4 is an SSA Logical Disk Drive and it is on the bus 04-06 (ignore the -L).

The **`lsdev -Cc adapter`** command with the bus number in Figure 5-12 on page 365 reveals the name and type of the disk adapter for hdisk4.

Note: At this point, it is still not possible to determine whether the hdisk is a real disk or a virtual disk. There may or may not be a one-to-one mapping between the hdisk and some real, physical disk. The method for determining that mapping depends upon the disk/adapter type.

Figure 5-12 on page 365 describes the method for an SSA disk.

Up to this point, the commands used did not require root privileges. However, subsequent commands may do so, as shown in Figure 5-12 on page 365.

The **`ssaraid`** command, with appropriate parameters, lists the physical disks (“pdisks”) comprising the SSA logical disk or hdisk. We used the **`ssaraid`** command in conjunction with other UNIX commands, as shown in Figure 5-12 on page 365, to limit the output to show just the pdisks for hdisk4 (for example, pdisk0, pdisk1, pdisk2, pdisk3, pdisk4, pdisk5 and pdisk6). Additional commands filtering the ssaraid output (not shown here) may be used to determine whether the pdisks are organized as RAID-0, RAID-5, and so on.

filemon

The **`filemon`** command uses the AIX trace facility to record information about disks, logical volumes and files. Use of this command can be processor-intensive.

In general, if you are trying to track down a disk performance problem, start with the broadest monitoring: at the disk level. **`Filemon`** tracing may begin immediately or be started later. In either case, the process needs to be explicitly stopped by using the **`trcstop`** command. The shell script shown in Figure 5-13 on page 367 runs **`filemon`** for three minutes, starting immediately.

```
#!/bin/sh

filemon -T 5000000 -P -v -u -o filemon.‘date +”%d%m”-”%H%M”‘ -O lv,lf
sleep 180
trcstop
```

Figure 5-13 Sample filemon script

Command options specify the size of the memory buffers holding trace data, the output file, and define what is traced. The script shown in traces logical volumes and files (**-lv, lf**).

If the **-u** option is omitted, the trace will only include files, volumes, and so on that are opened *after* the trace commenced. Since DB2 is likely to have opened some files beforehand, the **-u** option should always be used. If a file was opened prior to **filemon** starting, **filemon** will not be able to determine the filename; however, the inode number will be displayed. For command options, refer to the man page.

Filemon provides trace information down to the UNIX file level. For database objects, use the **db2 list tablespaces show detail** command and the **db2 list tablespace containers for <tsid> show detail** command to identify the underlying UNIX files and the related DB2 containers. Figure 5-14 on page 368 shows an edited version of the results of some **filemon** output.

The `Detailed File Stats` section in Figure 5-14 on page 368 shows the full path name of the file and the number of opens and seeks on the file in the sample period. The read and write statistics show the number, average, minimum, and maximum values. NaNQ represents “not a number”, and is typically caused by a divide by zero or other similar arithmetic operation.

The `Most Active Files` section in Figure 5-14 on page 368 shows active files ranked by throughput. The file name, inode, and volume are listed. Use the **df** command to map volumes to filesystems.

The `Most Active Logical Volumes` section in Figure 5-14 on page 368 shows active logical volumes, including raw and JFS log volumes, as well as those used for filesystems. If the volume corresponds to a filesystem, the description column shows the mount point for the filesystem.

```
Most Active Files
------------------------------------------------------------------------
  #MBs  #opns   #rds   #wrs  file                     volume:inode
------------------------------------------------------------------------
  19.2    703   4921      0  devices.cat              /dev/hd2:10408
   9.6      0   1227      7  pid=0_fd=17810
   4.8     21   1218      0  unix                     /dev/hd2:2362
.....
   0.1      4      8      2  SQLOGCTL.LFH             /dev/u1_lv:245775
.....
Most Active Logical Volumes
------------------------------------------------------------------------
  util  #rblk  #wblk   KB/s  volume                   description
------------------------------------------------------------------------
  0.00      0   1024    2.8  /dev/hd8                 jfslog
  0.00    480    840    3.6  /dev/hd2                 /usr
  0.00      0    456    1.2  /dev/hd4                 /
  0.00      0    182    0.5  /dev/hd1                 /home Frag_Sz.= 512
  0.00   1112    168    3.5  /dev/u1_lv               /u1
  0.00      0    153    0.4  /dev/hd9var              /var Frag_Sz.= 512

------------------------------------------------------------------------
Detailed File Stats
------------------------------------------------------------------------
.....
FILE: /u1/ezstor21/db2inst1/NODE0000/SQL00001/SQLOGCTL.LFH
   volume: /dev/u1_lv (/u1)  inode: 245775
opens:                  4
total bytes xfrd:       122880
reads:                  8       (0 errs)
  read sizes (bytes):   avg 12288.0 min   12288 max   12288 sdev    0.0
  read times (msec):    avg   0.026 min   0.020 max   0.048 sdev  0.009
writes:                         2       (0 errs)
  write sizes (bytes): avg 12288.0 min   12288 max   12288 sdev    0.0
  write times (msec):  avg    NaNQ min     INF max   0.000 sdev   NaNQ
lseeks:                         4
```

Figure 5-14 Sample filemon output

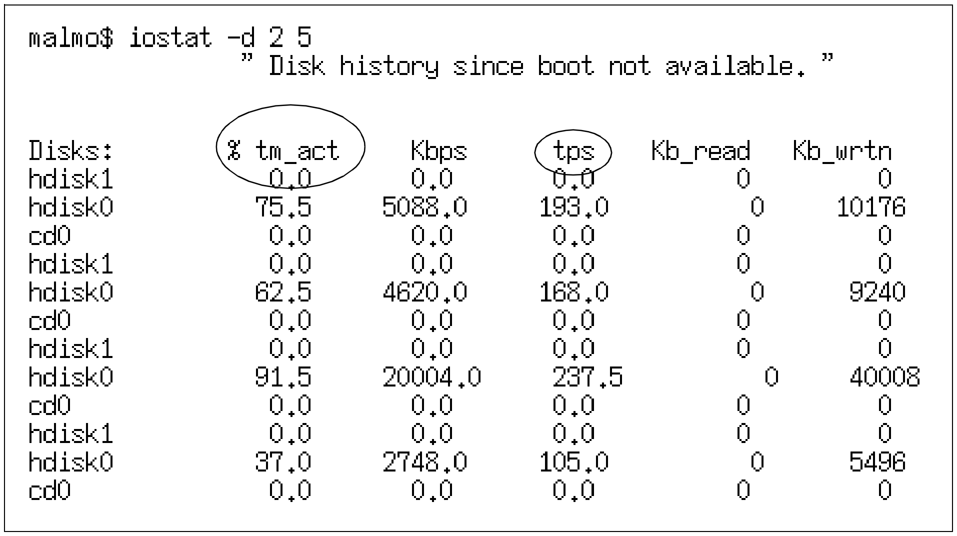
iostat

The **iostat** command displays statistics on disk, adapter, and system throughput. By default, the system does not collect detailed disk performance statistics, since this has a performance overhead. Reporting can be limited to a particular set of disks.

Figure 5-15 displays the result of the **iostat** command. The `Disk history since boot not available` display indicates that detailed statistics are not being collected. This can be changed by using the following command:

```
chdev -l sys0 -a iostat=true
```

The `% tm_act` column lists the percentage of time the disk was busy, representing bandwidth utilization, while the `tps` column shows the number of transfers per second to each disk.

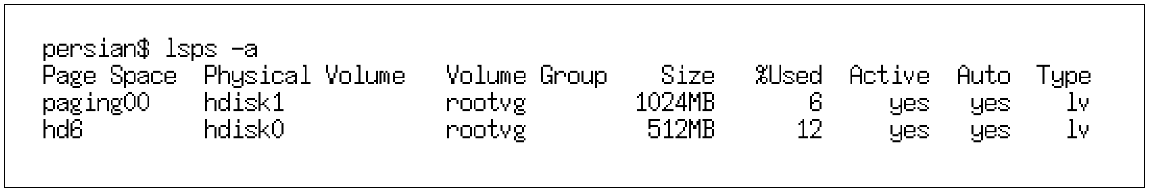


```
malmo$ iostat -d 2 5
               " Disk history since boot not available. "


Disks:        % tm_act     Kbps      tps    Kb_read   Kb_wrtn
hdisk1           0.0       0.0       0.0         0         0
hdisk0          75.5    5088.0     193.0          0     10176
cd0              0.0       0.0       0.0         0         0
hdisk1           0.0       0.0       0.0         0         0
hdisk0          62.5    4620.0     168.0          0      9240
cd0              0.0       0.0       0.0         0         0
hdisk1           0.0       0.0       0.0         0         0
hdisk0          91.5   20004.0      237.5          0     40008
cd0              0.0       0.0       0.0         0         0
hdisk1           0.0       0.0       0.0         0         0
hdisk0          37.0    2748.0     105.0          0      5496
cd0              0.0       0.0       0.0         0         0
```

Figure 5-15 iostat, 2-second sample interval

lsps

The **lsps** command displays information about the system paging space. It displays the size of the page files in MBytes, and the percent used. Figure 5-16 shows the results of this command.



```
persian$ lsps -a
Page Space  Physical Volume   Volume Group    Size   %Used  Active  Auto  Type
paging00    hdisk1            rootvg        1024MB       6     yes   yes    lv
hd6         hdisk0            rootvg         512MB      12     yes   yes    lv
```

Figure 5-16 lsps command showing paging space utilization

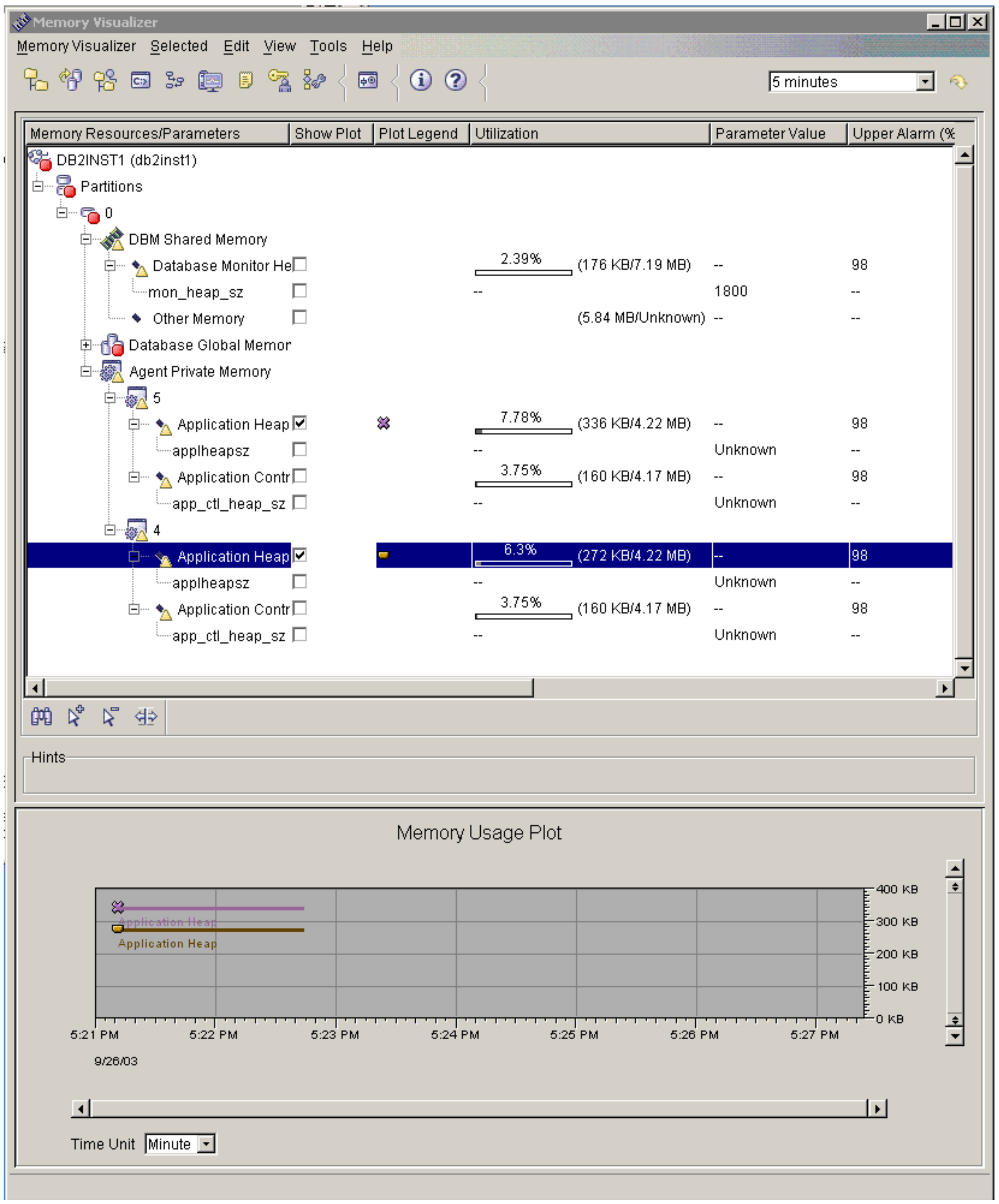
Memory Visualizer

The Memory Visualizer is a DB2 tool that helps database administrators to monitor the memory-related performance of an instance and all of its databases organized in a hierarchical tree. DBAs can drill down into specific performance areas to display values for the amount of memory allocated to the component and the current memory usage in the Memory Visualizer window.

In UNIX, open the Memory Visualizer with the **`db2memvis`** command. Figure 5-17 on page 371 shows a typical display.

The tool displays bar graphs showing current utilization of components of database global memory, database memory and application memory, along with configuration parameter values and alarm and threshold settings.

The user can elect to graph utilization of various values over time. This is shown in the lower pane of the display.



5 minutes

Memory Resources/Parameters	Show Plot	Plot Legend	Utilization	Parameter Value	Upper Alarm (%
DB2INST1 (db2inst1)					
Partitions					
0					
DBM Shared Memory					
Database Monitor He	☐		2.39% (176 KB/7.19 MB)	--	98
mon_heap_sz	☐		--	1800	--
Other Memory	☐		(5.84 MB/Unknown)	--	--
Database Global Memor					
Agent Private Memory					
5					
Application Heap	☑	✖	7.78% (336 KB/4.22 MB)	--	98
applheapsz	☐		--	Unknown	--
Application Contr	☐		3.75% (160 KB/4.17 MB)	--	98
app_ctl_heap_sz	☐		--	Unknown	--
4					
Application Heap	☑	▬	6.3% (272 KB/4.22 MB)	--	98
applheapsz	☐		--	Unknown	--
Application Contr	☐		3.75% (160 KB/4.17 MB)	--	98
app_ctl_heap_sz	☐		--	Unknown	--

Time Unit Minute

Figure 5-17 Memory Visualizer display

nmon

The **nmon** command is a public domain tool provided without warranty by IBM. It can be downloaded from the World Wide Web at the following site:

http://www.ibm.com/servers/esdd/articles/analyze_aix/agree_down.html

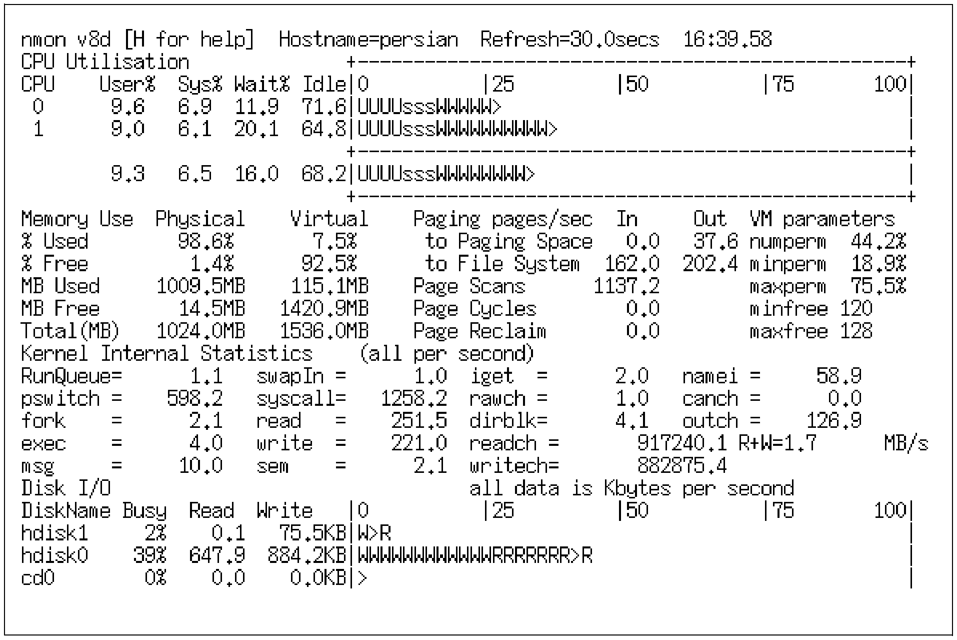
nmon can operate in interactive mode or in recording mode.

- In interactive mode, **nmon** displays a variety of system information such as CPU, disk, and memory utilization, various kernel counters, network throughput and so on. The screen update interval can be varied.
- In recording mode, **nmon** samples kernel counters at a specified interval and saves the data to a file in Microsoft® Excel .csv format.

  There is a second public domain tool that makes use of either Excel 2000 or Lotus® macros to analyze and graph the data. Additional information and a pointer to the download site can be found at:

  http://www.ibm.com/developerworks/eserver/articles/nmon_analyser/index.html

Figure 5-18 shows the results of the **nmon** command.



```
nmon v8d [H for help]  Hostname=persian  Refresh=30.0secs  16:39.58
CPU Utilisation       +--------------------------------------------------+
CPU     User%  Sys% Wait% Idle|0          |25          |50          |75       100|
 0       9.6   6.9  11.9  71.6|UUUUsssWWWWW>                                     |
 1       9.0   6.1  20.1  64.8|UUUUsssWWWWWWWWWWW>                               |
                              +--------------------------------------------------+
         9.3   6.5  16.0  68.2|UUUUsssWWWWWWWW>                                  |
                              +--------------------------------------------------+
Memory Use  Physical    Virtual    Paging pages/sec  In     Out  VM parameters
% Used        98.6%       7.5%      to Paging Space   0.0   37.6 numperm   44.2%
% Free         1.4%      92.5%      to File System  162.0  202.4 minperm   18.9%
MB Used     1009.5MB    115.1MB    Page Scans      1137.2        maxperm   75.5%
MB Free       14.5MB   1420.9MB    Page Cycles        0.0        minfree 120
Total(MB)   1024.0MB   1536.0MB    Page Reclaim       0.0        maxfree 128
Kernel Internal Statistics     (all per second)
RunQueue=       1.1   swapIn =      1.0  iget  =      2.0   namei =     58.9
pswitch =     598.2   syscall=   1258.2  rawch =      1.0   canch =      0.0
fork    =       2.1   read   =    251.5  dirblk=      4.1   outch =    126.9
exec    =       4.0   write  =    221.0  readch =       917240.1 R+W=1.7      MB/s
msg     =      10.0   sem    =      2.1  writech=       882875.4
Disk I/O                                 all data is Kbytes per second
DiskName Busy  Read  Write    |0          |25          |50          |75       100|
hdisk1     2%    0.1    75.5KB|W>R                                               |
hdisk0    39%  647.9   884.2KB|WWWWWWWWWWWWWWWRRRRRRRR>R                         |
cd0        0%    0.0     0.0KB|>                                                 |
```

Figure 5-18   nmon interactive display

The `CPU Utilization` section in Figure 5-18 displays CPU activity in numerical and graphical format.

The `Memory Use` section in Figure 5-18 on page 372 displays the amount of real and virtual memory used, plus paging rates and **vmtune** information. This output shows the system is using 98.6% of real memory, with 14.5 MB free. If this amount drops below **minfree** (120 4 KB pages = 490 KB), then the system will aggressively begin freeing up memory.

The paging rates section in Figure 5-18 on page 372 shows the number of pages per second moved to and from both the paging system and the filesystems—and this is an important distinction. Filesystem paging represents data being read from or written to files in the filesystem, and is therefore “normal”. Paging to and from the paging space should ideally be zero.

The `Kernel Internal Statistics` section in Figure 5-18 on page 372 show a number of useful counters. The `RunQueue` represents the number of processes waiting to run on a CPU; this value should ideally be zero.

The `Disk I/O` section in Figure 5-18 on page 372 shows disk utilization. If there are a large number of disks, try using the `Adapter I/O` display instead (**a** option).

ps

The **ps** command displays information about processes. It can be used to identify CPU-intensive processes. The **ps** command examines data in kernel memory, which may be in a state of change. It gathers data on a per process basis. Note that information on process sizes does not account for sharing of virtual memory between processes. The data displayed in the `%MEM`, `SIZE`, `TSIZ` and `RSS` columns do not reflect actual memory usage!

On AIX, there are two flavors of output: the X/Open standard, and the Berkeley standard.

- X/Open output is displayed when command options with a hyphen (-) are used.
- Berkeley options do not use a hyphen.

When no options are given, basic information about the current user’s processes is displayed. This display is common to both formats.

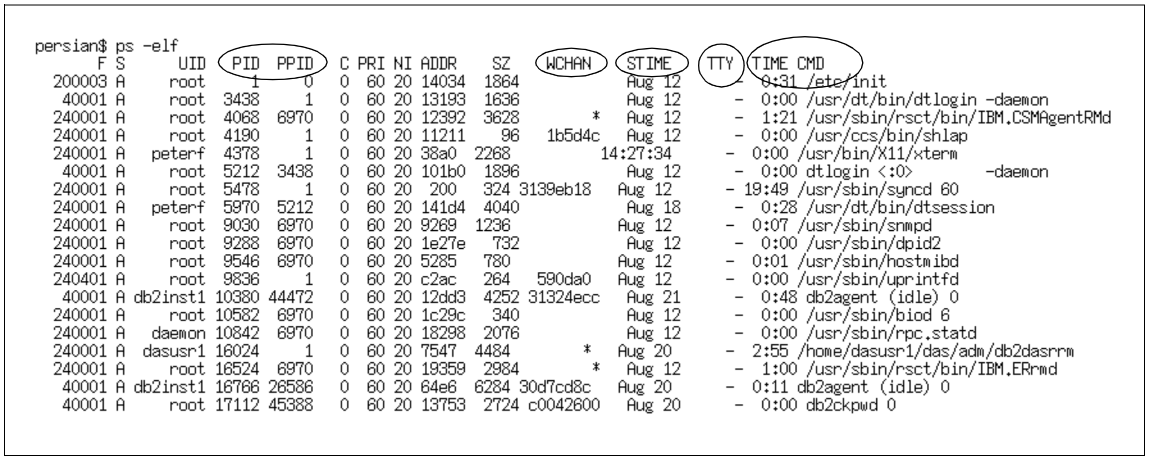
CPU-intensive processes may be identified as follows:

- The C column (**-f**, **-l**, **l**) indicates the amount of time recently used by the process in clock ticks. Look for a high number relative to other processes.
- The TIME column (all flags) shows the total time used by the process since it started. Look for large values that are also increasing.
- The %CPU column (**u**, **v**) represents the time used by the process since it started, divided by the elapsed time since the process started.

The two most useful option combinations are **`-elf`** as shown in Figure 5-19, and **`aux`** as shown in Figure 5-20 on page 375.

Figure 5-19 shows the output of the **`ps`** command with options **`-elf`**. The columns of most interest are as follows:

- `UID` - the user executing the command.
- `PID` - the process ID of the process.
- `PPID` - the PID of the process's parent.
- `WCHAN` - if not blank, this corresponds to an address in the kernel at which the process is suspended waiting for something (such as an I/O) to complete.
- `STIME` - the time (or date) the process started.
- `TTY` - the terminal the process was started on.
- `TIME` - minutes and seconds of CPU time.
- `CMD` - the command and its parameters. The amount displayed depends on what **`ps`** finds in certain kernel data structures.



```
persian$ ps -elf
       F S      UID   PID  PPID   C PRI NI ADDR    SZ    WCHAN    STIME    TTY  TIME CMD
  200003 A     root     1     0   0  60 20 14034  1864             Aug 12      -  0:31 /etc/init
   40001 A     root  3438     1   0  60 20 13193  1636             Aug 12      -  0:00 /usr/dt/bin/dtlogin -daemon
  240001 A     root  4068  6970   0  60 20 12392  3628         *   Aug 12      -  1:21 /usr/sbin/rsct/bin/IBM.CSMAgentRMd
  240001 A     root  4190     1   0  60 20 11211    96   1b5d4c   Aug 12      -  0:00 /usr/ccs/bin/shlap
  240001 A   peterf  4378     1   0  60 20 38a0  2268          14:27:34     -  0:00 /usr/bin/X11/xterm
   40001 A     root  5212  3438   0  60 20 101b0  1896             Aug 12      -  0:00 dtlogin <:0>        -daemon
  240001 A     root  5478     1   0  60 20  200   324 3139eb18   Aug 12     - 19:49 /usr/sbin/syncd 60
  240001 A   peterf  5970  5212   0  60 20 141d4  4040             Aug 18      -  0:28 /usr/dt/bin/dtsession
  240001 A     root  9030  6970   0  60 20 9269  1236             Aug 12     -  0:07 /usr/sbin/snmpd
  240001 A     root  9288  6970   0  60 20 1e27e   732             Aug 12      -  0:00 /usr/sbin/dpid2
  240001 A     root  9546  6970   0  60 20 5285   780             Aug 12     -  0:01 /usr/sbin/hostmibd
  240401 A     root  9836     1   0  60 20 c2ac   264   590da0   Aug 12     -  0:00 /usr/sbin/uprintfd
   40001 A db2inst1 10380 44472   0  60 20 12dd3  4252 31324ecc    Aug 21      -  0:48 db2agent (idle) 0
  240001 A     root 10582  6970   0  60 20 1c29c   340             Aug 12      -  0:00 /usr/sbin/biod 6
  240001 A   daemon 10842  6970   0  60 20 18298  2076             Aug 12      -  0:00 /usr/sbin/rpc.statd
  240001 A  dasusr1 16024     1   0  60 20 7547  4484         *   Aug 20     -  2:55 /home/dasusr1/das/adm/db2dasrrm
  240001 A     root 16524  6970   0  60 20 19359  2984         *   Aug 12      -  1:00 /usr/sbin/rsct/bin/IBM.ERrmd
   40001 A db2inst1 16766 26586   0  60 20 64e6  6284 30d7cd8c   Aug 20     -  0:11 db2agent (idle) 0
   40001 A     root 17112 45388   0  60 20 13753  2724 c0042600   Aug 20      -  0:00 db2ckpwd 0
```

Figure 5-19 ps -elf command

```
persian$ ps aux
USER       PID %CPU %MEM    SZ  RSS    TTY STAT    STIME  TIME COMMAND
root       516 49.1  2.0    12 12940      - A      Aug 12 17163:10 wait
root       774 49.0  2.0    12 12940      - A      Aug 12 17115:24 wait
db2inst1 48794  0.1  0.0 6236 1388      - A      Aug 21 12:53 db2agent (idle) 0
root      5478  0.1  0.0  324  188      - A      Aug 12 19:49 /usr/sbin/syncd 6
dasusr1  16024  0.0  0.0 4484  888      - A      Aug 20  2:55 /home/dasusr1/das
db2inst1 39118  0.0  0.0 6208  456      - A      Aug 21  1:57 db2agent (idle) 0
db2inst1 35558  0.0  0.0 4780  280      - A      Aug 20  2:26 db2agent (idle) 0
db2inst1 31722  0.0  0.0 2572  172      - A      Aug 20  2:18 db2pfchr 0
root         0  0.0  2.0    16 12944      - A      Aug 12  6:32 swapper
root     13678  0.0  0.0 2216  528      - A      Aug 12  5:52 /usr/opt/db2_08_0
db2inst1 51438  0.0  0.0 2580  100      - A      Aug 20  1:56 db2pfchr 0
db2inst1 27752  0.0  0.0 7016 1224      - A      Aug 21  1:14 db2agent (idle) 0
db2inst1 19614  0.0  0.0 6924  888      - A      Aug 20  1:18 db2agent (idle) 0
db2inst1 40228  0.0  0.0 2500  112      - A      Aug 20  1:13 db2pclnr 0
....
```
Figure 5-20 ps aux example

Most of the columns in Figure 5-19 on page 374 and Figure 5-20 are the same; however, the additional `%CPU` column in Figure 5-20 shows the percentage of time that the process used the CPU since it started. It is calculated by dividing the CPU time used by the elapsed time. The wait processes in Figure 5-20 are system idle processes used to account for any time that the system is not running user processes.

Figure 5-21 shows the output of the **ps** command with the options **avx**. Here again, the output is similar, except for the additional `PGIN` column, which displays the number of page faults that resulted in disk I/Os for this process.

```
persian$ ps avx
   PID    TTY STAT   TIME PGIN  SIZE   RSS   LIM  TSIZ   TRS %CPU %MEM COMMAND
     0      - A      6:32    7    16 12944    xx     0 12928  0.0  2.0 swapper
     1      - A      0:31   75  1840   124 32768    26    32  0.0  0.0 /etc/init
   516      - A  17162:58    0    12 12940    xx     0 12928 49.1  2.0 wait
   774      - A  17115:13    0    12 12940    xx     0 12928 49.0  2.0 wait
....
 16766      - A      0:11  685  6220   428    xx    65    12  0.0  0.0 db2agent
 17112      - A      0:00   67  2660   272    xx    65    12  0.0  0.0 db2ckpwd
 17234      - A      0:03   82  2020    72 32768   340    32  0.0  0.0 /usr/HTTP
 17840      - A      0:33 37158  4376   520    xx    65    12  0.0  0.0 db2agent
 18018      - A      0:00    0  2496    28    xx    65    12  0.0  0.0 db2pclnr
 18536      - A      0:24  389 11984  1520    xx    65    12  0.0  0.0 db2agent
 18934      - A      0:00  189  3800   124 32768   588     0  0.0  0.0 /usr/dt/b
 19310      - A      0:00    8  2568    48    xx    65    12  0.0  0.0 db2pfchr
 19614      - A      1:18 118623  6860   880    xx    65    12  0.0  0.0 db2agent
....
```
Figure 5-21 ps avx example

sar

System activity recorder (**sar**) is common to all flavors of UNIX, and is therefore a popular performance tool in multivendor environments.

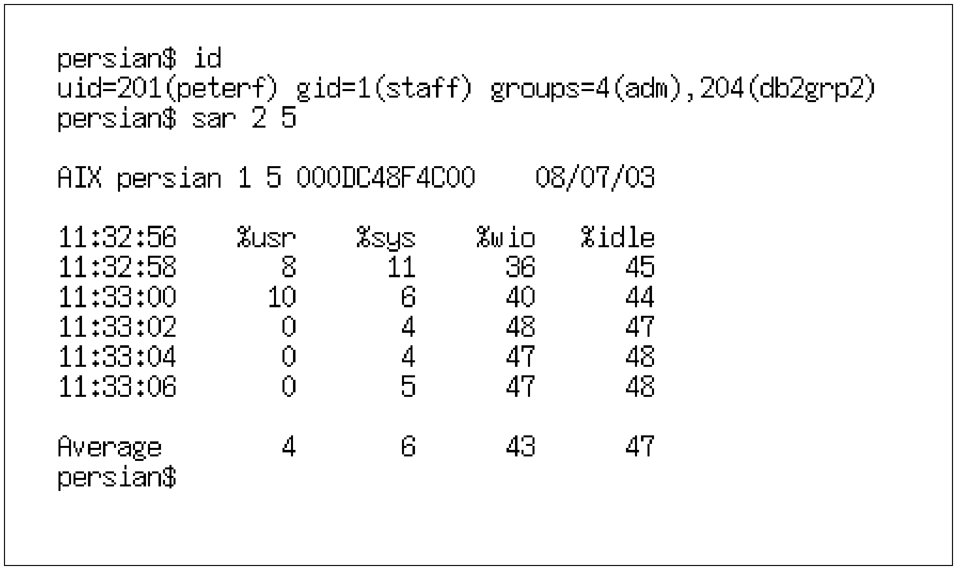
> **Note:** **sar** is not installed by default in AIX 4.3.3 or AIX 5.1, and can be installed from the bos.acct package. It is however, installed by default in AIX 5.2.

sar records a large number of performance metrics, though some of them are not useful or meaningful in AIX due to differences in its implementation. Check the man pages for these specific details.

A standard script (crontab) is available that invokes **sar** to record system metrics every 20 minutes from 08:00 to 17:00 Monday through Friday, and every hour outside these times. This script has the entries commented out by default. If you turn this collection on, **sar** retains the data for one week.

sar can be used to generate performance metrics for a particular item for a number of samples at a specified interval by using the appropriate command options. **sar** will also display metrics from a previously saved sar file.

Figure 5-22 shows sample output of the **sar** command.



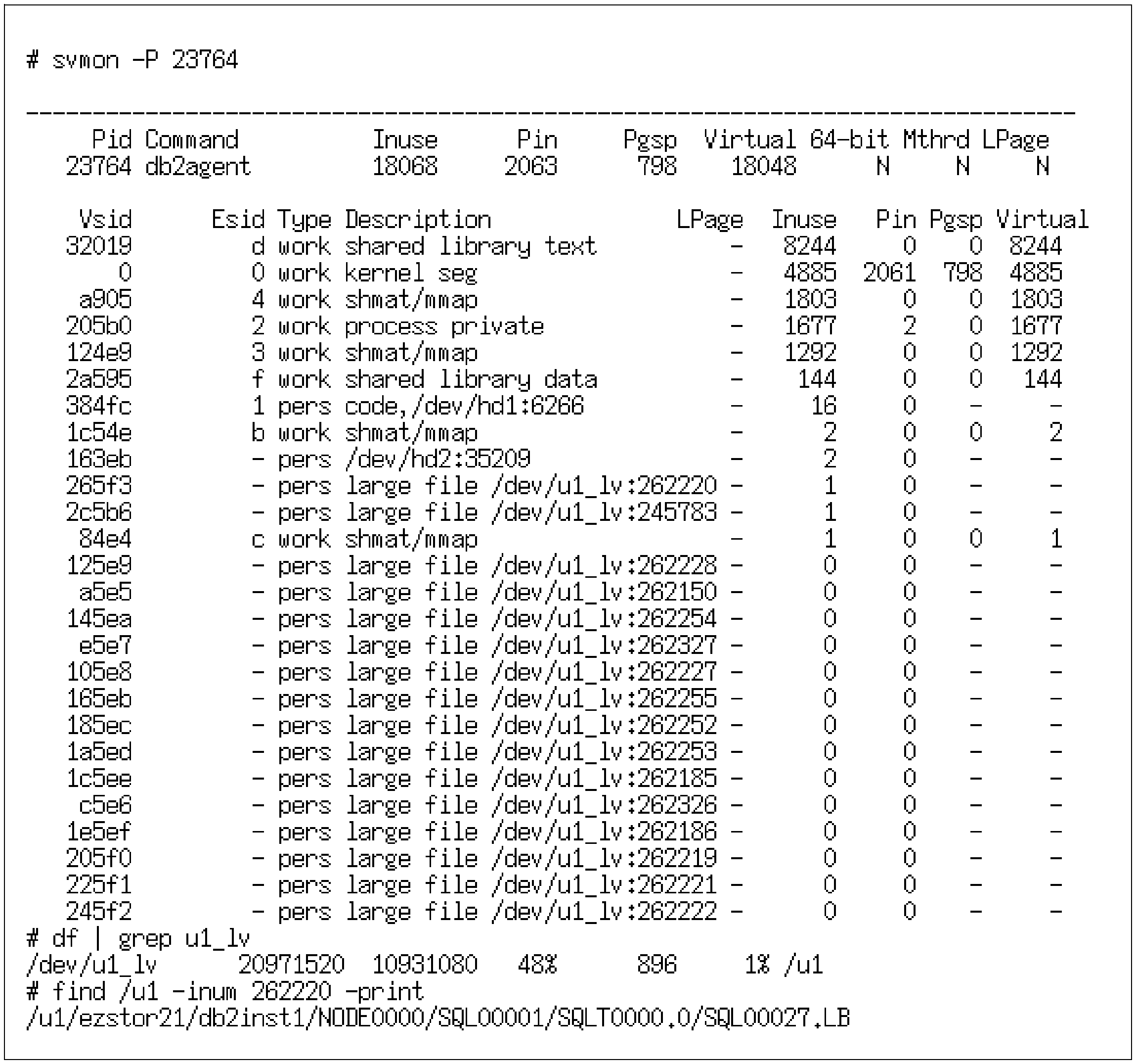
```
persian$ id
uid=201(peterf) gid=1(staff) groups=4(adm),204(db2grp2)
persian$ sar 2 5

AIX persian 1 5 000DC48F4C00    08/07/03

11:32:56    %usr    %sys    %wio   %idle
11:32:58       8      11      36      45
11:33:00      10       6      40      44
11:33:02       0       4      48      47
11:33:04       0       4      47      48
11:33:06       0       5      47      48

Average        4       6      43      47
persian$
```

Figure 5-22 sar default output

svmon

svmon displays various memory statistics for an individual process or selected processes, and is used to discover users/processes consuming memory.

Figure 5-23 displays the output of the **svmon** command for an individual process identified by its PID.



```
# svmon -P 23764

-------------------------------------------------------------------------------
     Pid Command          Inuse      Pin     Pgsp  Virtual 64-bit Mthrd LPage
   23764 db2agent         18068     2063      798    18048      N     N     N

    Vsid      Esid Type Description              LPage  Inuse   Pin Pgsp Virtual
   32019         d work shared library text          -   8244     0    0  8244
       0         0 work kernel seg                   -   4885  2061  798  4885
    a905         4 work shmat/mmap                   -   1803     0    0  1803
   205b0         2 work process private              -   1677     2    0  1677
   124e9         3 work shmat/mmap                   -   1292     0    0  1292
   2a595         f work shared library data          -    144     0    0   144
   384fc         1 pers code,/dev/hd1:6266           -     16     0    -     -
   1c54e         b work shmat/mmap                   -      2     0    0     2
   163eb         - pers /dev/hd2:35209               -      2     0    -     -
   265f3         - pers large file /dev/u1_lv:262220 -      1     0    -     -
   2c5b6         - pers large file /dev/u1_lv:245783 -      1     0    -     -
    84e4         c work shmat/mmap                   -      1     0    0     1
   125e9         - pers large file /dev/u1_lv:262228 -      0     0    -     -
    a5e5         - pers large file /dev/u1_lv:262150 -      0     0    -     -
   145ea         - pers large file /dev/u1_lv:262254 -      0     0    -     -
    e5e7         - pers large file /dev/u1_lv:262327 -      0     0    -     -
   105e8         - pers large file /dev/u1_lv:262227 -      0     0    -     -
   165eb         - pers large file /dev/u1_lv:262255 -      0     0    -     -
   185ec         - pers large file /dev/u1_lv:262252 -      0     0    -     -
   1a5ed         - pers large file /dev/u1_lv:262253 -      0     0    -     -
   1c5ee         - pers large file /dev/u1_lv:262185 -      0     0    -     -
    c5e6         - pers large file /dev/u1_lv:262326 -      0     0    -     -
   1e5ef         - pers large file /dev/u1_lv:262186 -      0     0    -     -
   205f0         - pers large file /dev/u1_lv:262219 -      0     0    -     -
   225f1         - pers large file /dev/u1_lv:262221 -      0     0    -     -
   245f2         - pers large file /dev/u1_lv:262222 -      0     0    -     -
# df | grep u1_lv
/dev/u1_lv      20971520  10931080   48%      896     1% /u1
# find /u1 -inum 262220 -print
/u1/ezstor21/db2inst1/NODE0000/SQL00001/SQLT0000.0/SQL00027.LB
```

Figure 5-23 Output of svmon -P command

The first two lines in Figure 5-23 on page 377 show the process id and command name followed by the number of memory pages used, amount of pinned memory, the number of pages of paging space allocated, whether the process is 64-bit, and whether it is multi-threaded.

The remainder of the output shows memory segment usage. The columns are listed and explained in Table 5-5.

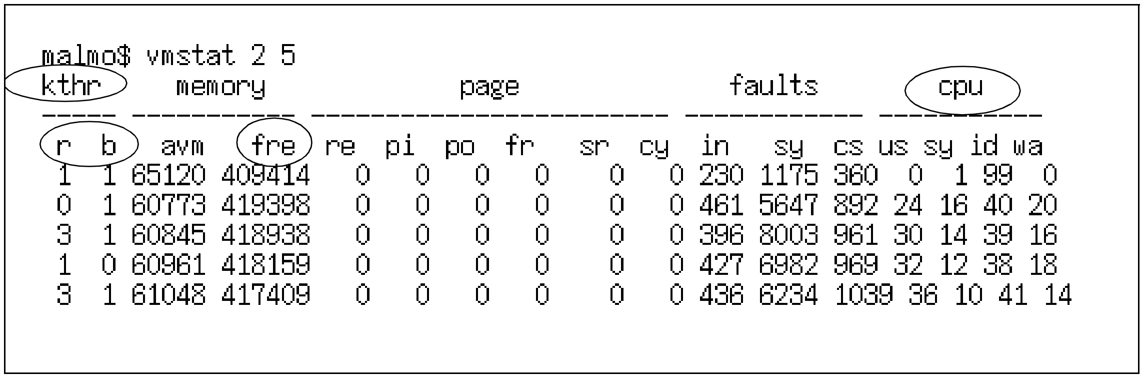
Table 5-5 svmon output

Column name	**Meaning**
Vsid	Virtual segment id.
Esid	Effective segment id, which corresponds to the segment register number in hexadecimal.
Type	work - part of the program's code or data pers - a file There are some additional types.
Description	Depends on the type. For a pers segment, this is the filesystem's logical volume and the file inode number. You can use this to find the path name of the file as shown at the bottom of the figure.
In use	Currently, the number of pages in real memory for this segment.
Pin	The number of pages pinned in this segment.
Pgsp	The number of pages used in paging space for this segment. Only relevant for working segments.
Virtual	Number of pages allocated for the virtual space tor the segment.

vmstat

The `vmstat` command displays statistics on memory usage, system load, and run/blocked process queues.

Figure 5-24 on page 379 shows the output of **`vmstat`**.



```
malmo$ vmstat 2 5
kthr     memory             page              faults        cpu
----- ----------- ------------------------ ------------ -----------
 r  b   avm   fre  re  pi  po  fr   sr  cy  in   sy  cs us sy id wa
 1  1 65120 409414  0   0   0   0    0   0 230 1175 360  0  1 99  0
 0  1 60773 419398  0   0   0   0    0   0 461 5647 892 24 16 40 20
 3  1 60845 418938  0   0   0   0    0   0 396 8003 961 30 14 39 16
 1  0 60961 418159  0   0   0   0    0   0 427 6982 969 32 12 38 18
 3  1 61048 417409  0   0   0   0    0   0 436 6234 1039 36 10 41 14
```

Figure 5-24 vmstat, 2-second sample interval

When using **vmstat** in interval mode, ignore the first row of values because some counters represent cumulative values since system boot.

The `kthr` columns labelled `r` and `b` show the number of threads placed in the relevant queue per second; that is, the run queue (**r**) and the wait queue (**w**). Processes in the wait queue are blocked and waiting on some resource (for example, IO completion).

The `cpu` columns show the percentages of user, system, idle and waitIO time.

If the number of memory pages in the `fre` column gets below a certain value (see **vmtune**), the system starts paging out memory to get more free pages.

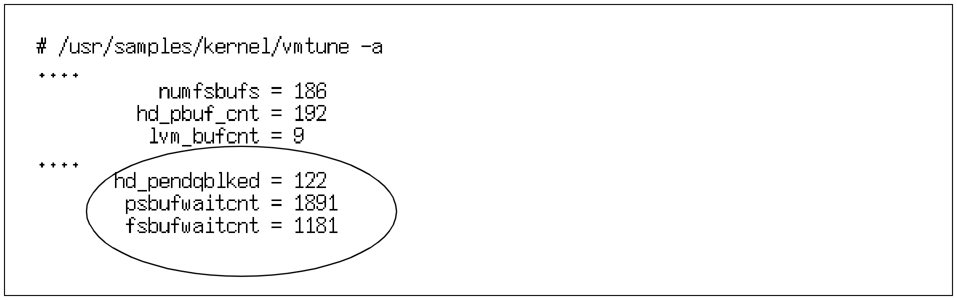
vmtune

vmtune displays or dynamically modifies various kernel parameters associated with the virtual memory subsystem. To preserve settings across reboots, certain steps need to be taken.

Note: In AIX 5.2, the **vmtune** has been replaced by the commands **vmo** and **ioo**. A consistent method has also been provided for preserving settings across reboots.

vmtune is *not* installed by default during operating system installation, but it can be found in the bos.adt.samples fileset. **vmtune** is located in the /usr/samples/kernel directory.

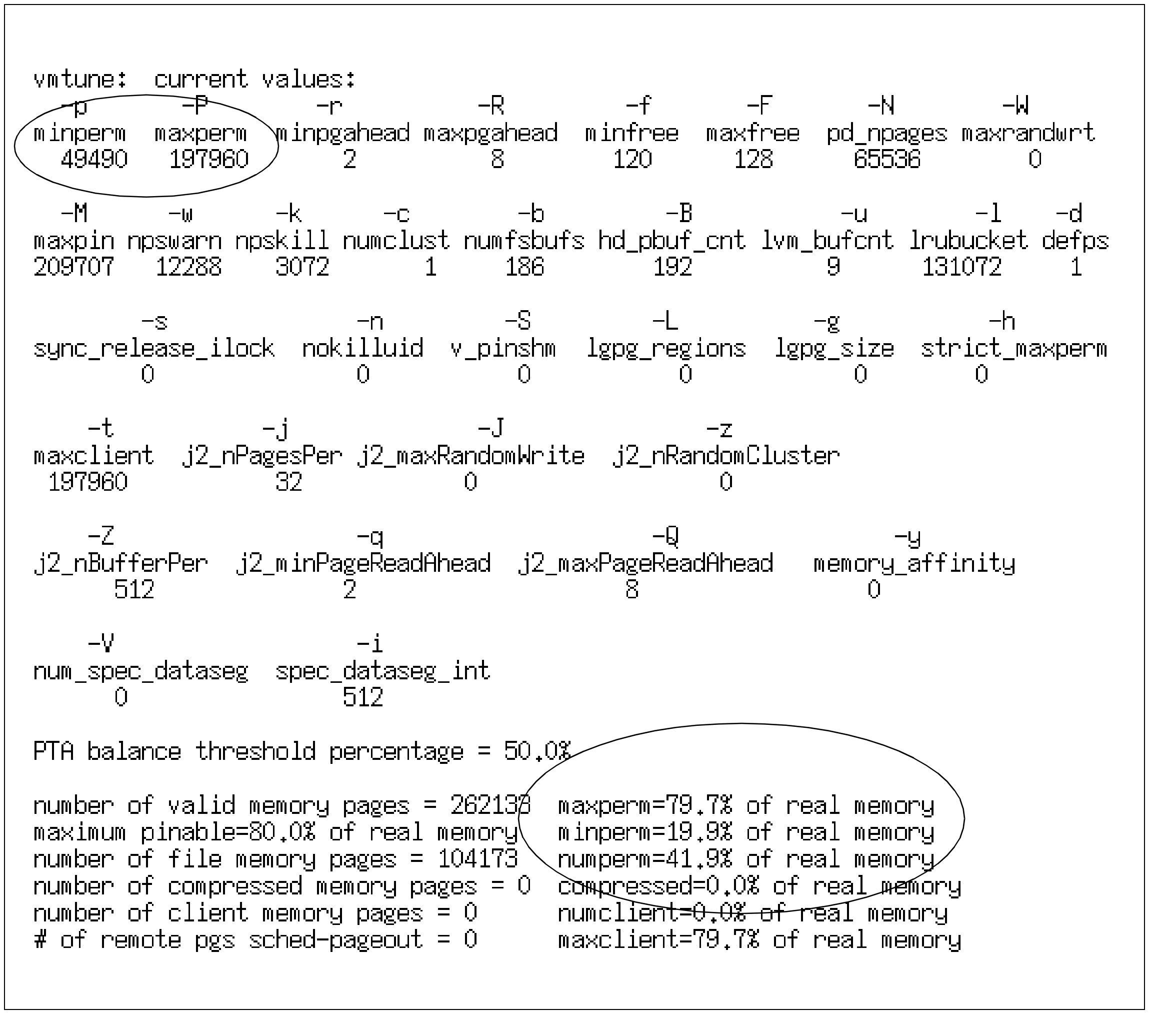
Figure 5-25 on page 380 displays the various operating system variables and their values in column format; included are some counters useful in tuning buffers.



```
# /usr/samples/kernel/vmtune -a
....
          numfsbufs = 186
        hd_pbuf_cnt = 192
         lvm_bufcnt = 9
....
      hd_pendqblked = 122
       psbufwaitcnt = 1891
       fsbufwaitcnt = 1181
```

Figure 5-25 Extract from vmtune -a output

Figure 5-26 on page 381 displays the results of the **vmtune** command where values of several operating system variables are expressed as numbers of 4 KB pages and percentages.



```
vmtune:  current values:
  -p       -P        -r          -R         -f       -F       -N        -W
minperm  maxperm  minpgahead maxpgahead  minfree  maxfree  pd_npages maxrandwrt
  49490   197960      2          8         120      128     65536        0

  -M      -w      -k      -c        -b         -B          -u        -l    -d
maxpin npswarn npskill numclust numfsbufs hd_pbuf_cnt lvm_bufcnt lrubucket defps
209707   12288    3072       1      186        192          9      131072     1

        -s              -n         -S          -L           -g           -h
sync_release_ilock  nokilluid  v_pinshm  lgpg_regions  lgpg_size  strict_maxperm
        0               0          0           0            0         0

    -t           -j            -J               -z
maxclient  j2_nPagesPer j2_maxRandomWrite  j2_nRandomCluster
 197960          32           0                 0

    -Z               -q                   -Q                 -y
j2_nBufferPer  j2_minPageReadAhead  j2_maxPageReadAhead   memory_affinity
      512              2                  8                   0

    -V                  -i
num_spec_dataseg  spec_dataseg_int
      0                512

PTA balance threshold percentage = 50.0%

number of valid memory pages = 262138  maxperm=79.7% of real memory
maximum pinable=80.0% of real memory   minperm=19.9% of real memory
number of file memory pages = 104173   numperm=41.9% of real memory
number of compressed memory pages = 0  compressed=0.0% of real memory
number of client memory pages = 0      numclient=0.0% of real memory
# of remote pgs sched-pageout = 0      maxclient=79.7% of real memory
```

Figure 5-26 vmtune default output

5.3 Windows platform

Windows 2000 is Microsoft's multiuser operating system on Intel® servers.

As with AIX, we briefly discuss the following key performance drivers:

- Operating system considerations
- Memory considerations
- Disk and filesystem considerations

Again as with the AIX platform, best practices and recommendations are provided for each of these performance drivers, and are classified as being general and DB2-specific. Typical performance monitoring tools for these performance drivers are also described.

5.3.1 Operating system considerations

In this section, we provide the following:

1. Windows review
2. General performance recommendations
3. DB2-specific performance recommendations

Windows review

Current implementations of the Windows operating system are based on 32-bit hardware (IA-32) and use 3-bit addressing. These implementations include Windows 2000, Windows XP, Windows Advanced Server, and Windows Datacenter Server. Most hardware platforms implement the IA-32 Physical Address Extension architecture, which can address up to 64 GB of memory; see Table 5-6.

Note the 64-bit implementations exist for the Datacenter editions of the WIndows operating system, and they require corresponding 64-bit Intel or IA-64-compatible hardware.

Table 5-6 Windows operating system characteristics

Feature	Windows 2000/XP	Advanced Server	Datacenter
CPU	4	8	32
Memory (GBytes)	4	8	64

Windows 2003 operating systems versions support larger amounts of memory.

Both 32-bit and 64-bit implementations of DB2 are available on Windows. Unlike the AIX environment, the DB2 bit size *must* match the Windows operating system bit size.

The DB2 process model on Windows differs from that on UNIX systems in that DB2 is implemented as a single process, multi-threaded application. The server process is called db2sysc. Other processes exist for the administration server (**db2dasrrm**), license management (**db2licd**), and so on.

A Windows NT (and Windows 2000) 32-bit process has a 4 GB address space. This is normally split into 2 GB of kernel address space and 2 GB of user address space. Hardware address protection mechanisms prevent the user process instructions from accessing kernel space.

Support for more than 4 GBytes of memory requires one or more startup or boot options in addition to DB2 configuration. These include the /PAE and /3GB boot options.

The /3GB option changes the process address boundary so that the user address space is 3 GB and the kernel address space is 1 GB. However, /3GB cannot be used if you want to use more than 16 GBytes of memory. The /PAE option enables access to memory beyond 4 GB.

General performance recommendations

- Add the following useful columns to your Task Manager: Memory Usage, Memory Usage Delta, I/O Reads, I/O Read Bytes, I/O Writes, I/O Write Bytes. See the Tools section for a description of these columns.

DB2-specific recommendations

- Following the installation of DB2, check that the DB2 performance counters are visible from Performance Monitor.
- To take advantage of memory greater than 4 GBytes using the Address Windows Extensions (AWE) feature, the DB2 administrator must have the “lock pages in memory” user privilege.
- To increase the user address space of a Windows process from 2 Gbytes to 3 Gbytes, set the /3GB boot option. Do this by editing the boot.ini file and adding /3GB to the end of the line defining the boot image. For example:

  ```
  multi(0)disk(0)rdisk(0)partition(1)\WINNT="Microsoft Windows 2000
  Professional" /fastdetect /3GB
  ```

- If you intend to use AWE to access memory beyond 4 GBytes, add the /PAE option to the boot.ini entry as above.

5.3.2 Memory considerations

In this section, we provide the following:

1. Windows virtual memory architecture review
2. General performance recommendations
3. DB2-specific performance recommendations

Windows virtual memory architecture review

The DB2 implementation on Windows uses a threaded model: there is only one process, and all threads can access the process's address space. Therefore, there is no equivalent of the UNIX shared memory model.

Windows 2000 uses a virtual memory management subsystem similar to UNIX systems. The paging space file is called pagefile.sys and is normally found in the root directory of the C: drive. Additional paging space may be allocated as required.

You can determine the current paging file allocation via **My Computer** -> **System Properties** -> **Performance Options** -> **Change**. This displays the `Virtual Memory` panel showing paging space allocation on drives, and allows you to change the size of the paging space files. Setting the Initial Size and Maximum Size values to the same value will prevent fragmentation.

You can use DB2 Memory Visualizer to show memory consumption for database shared and global memory and agent private heaps.

Windows 32-bit processes can access memory above the 4 Gbyte boundary imposed by a 32-bit hardware address pointer by using the Microsoft Address Windows Extensions (AWE). DB2 can address the memory above 4 GB by setting the **`DB2_AWE`** registry variable, and can utilize memory up to 64 GBytes for buffer pools.

DB2 makes use of a "window" within the process's user address space to access memory above 4 GBytes. The size of the window is configurable and must be between two buffer pool pages and 1.5 GBytes (or 2.5 GBytes, when the /3GB option is used).

Memory accessed using AWE is associated with one or more buffer pools. The buffer pool must be defined (that is, it must exist in SYSCAT.BUFFERPOOLS) before it can be used by AWE. The **`DB2_AWE`** registry variable is used to define the buffer pool, the number of pages in the buffer pool, and the size of the window. Memory accessed via AWE is also "locked'; that is, it will not get paged out. This can provide a performance benefit—but be aware that it may affect the performance of other, non-DB2 processes. The format of the registry variable setting is as follows:

```
db2set DB2_AWE=<buffer_pool_id>,number_of_pages, size_of_window
```

Where **`buffer_pool_id`** is the identifier for the buffer pool. This can be found in the `BUFFERPOOLID` column of the relevant row in SYSCAT.BUFFERPOOLS. The buffer pool must have been defined before you define the **`DB2_AWE`** variable entry.

The **number_of_pages** represents the number of pages of “buffer pool page size” in the memory region, and **size_of_window** represents the window size in “buffer pool page size” pages. Thus the region size is: **(number_of_pages+size_of_window)*buffer_pool_page_size** bytes.

To define multiple regions, add semi-colon-separated entries, as shown in the following example:

```
db2set DB2_AWE=1,100,20;2,500,100
```

If AWE support is enabled, extended storage cannot be used for any buffer pools.

General performance recommendations

- Avoid placing paging files on system disks or heavily used disks, if possible. The total paging space should typically be 1.5 times the amount of memory on the system.
- Windows 2000 allocates paging space when the process allocates memory. Therefore, paging space in pagefile.sys must be allocated for all potential memory used by processes including DB2. Size the space required based on the values of DB2 configuration variables.
- Monitor page faulting via **Task Manager** -> **Processes/PF Delta** or via Performance Monitor and the Memory object. Using Performance Monitor, you can graph Page Faults/sec. You may wish to add Pages Input/sec and Pages Output/sec to the graph. Ensure you have a baseline against which to compare abnormal performance.

  As for AIX, page fault rates should ideally be zero, although a low number (for example, between 10 and 20 page faults/second) is acceptable.

DB2-specific recommendations

We recommend the following best practices for DB2 on Windows:

1. If your system never or rarely pages and you have a great deal of memory and paging space, set **DB2MEMDISCLAIM=NO**; otherwise, set **DB2MEMDISCLAIM=YES** and set **DB2MEMMAXFREE=8000000**. DB2 registry variables are set via the **db2set** command.
2. If the system is used exclusively for DB2 and ultimate performance is required, consider defining buffer pool access via AWE (locking buffer pool pages in memory).
3. If you have more than 4 GBytes of memory and can dedicate additional memory to DB2, use AWE to define memory regions to access one or more buffer pools in the space above 4 GBytes.

5.3.3 Disk and filesystem considerations

In this section, we provide the following:

- Disk subsystem overview
- Disk subsystem recommendations
- Filesystem recommendations
- DB2-specific recommendations

Disk subsystem overview

The observations about disk subsystem design in the AIX environment are also relevant to larger Windows 2000 systems. However, smaller Windows 2000 systems are more likely to be configured with fewer adapters and disks. SCSI RAID adapters are common in the Windows 2000 marketplace, and care is required in laying out databases on these disk subsystems. Give careful consideration to the physical placement of data and the potential for disk access contention when files share disks.

Windows 2000 buffers filesystem reads in a similar manner to AIX. However, this behavior can be modified by the **DB2NTNOCACHE** variable. The default value is OFF. In this case, Windows caches all files (except those containing LOBs). If the variable is set to ON, filesystem caching is eliminated allowing more memory for buffer pools, sort heaps, and so on.

The variable **DB2BPVARS** can be used to set two parameters that can have an impact on filesystem performance:

- **NO_NT_SCATTER** can only be used when **DB2NTNOCACHE** is **ON**. When **NO_NT_SCATTER** is set to the value **1**, DB2 will used “scattered reads” to read pages into the buffer pool.
- **NUMPREFETCHQUEUES** (default = 1) may be set to a number between 1 and **NUM_IOSERVERS**. It may be advantageous to increase **NUMPREFETCHQUEUES** on systems with multiple CPUs and disk adapters. This will require experimentation to determine the best values.

Altering these variables should be approached with caution, and subsequent results should be verified with benchmarks.

The filemon tool from Sysinternals (see “filemon” on page 389) captures information about access to files. It is capable of generating very large amounts of output and, unless you have a multi-processor system, it may perturb performance measurements. However, it will capture information about reads and writes to individual files. This may help narrow the focus when attempting to diagnose an IO performance problem.

Disk subsystem recommendations

- Review the recommendations made in the AIX section (5.2.3, “Disk and filesystem considerations” on page 346). Many of them also apply in Windows environments.

Filesystem recommendations

The native filesystem for Windows 2000 is NTFS. It provides the ability to create large files. There are several features which can be configured to improve performance:

- Use a large cluster size. The cluster size is the unit of space allocation and can vary from 512 bytes to 64 KB. Use a larger size to enable larger files to be created.

  Note: The cluster size is set when the filesystem is created and cannot be changed.

- Do not use file compression, because it requires additional processing overhead. If additional storage is required, invest in more disks.
- Disable Last Access time stamp. As Windows traverses directories (folders), it updates the Last Accessed time stamp costing unnecessary disk IOs. Disable this feature by using the Registry Editor to select **`HKEY_LOCAL_MACHINE->System->CurrentControlSet->Control->Filesystems`** and add a key named **`NtfsDisableLastAccessUpdate`** of type **`REG_DWORD`** with a value of **`1`**.

DB2-specific recommendations

- If you use DMS file containers, set **`DB2NTNOCACHE=ON`** to eliminate filesystem caching. This prevents double data movement and also allows you to assign more memory to buffer pools.

5.3.4 Monitoring and problem determination tools

DB2 UDB’s port to the Windows platform is well integrated with common Windows utilities such as Event Manager and Performance Manager.

Microsoft provides several performance monitoring tools with Windows 2000. Additional tools and information are available in the Windows 2000 Resource Kit at the following URL:

`http://www.microsoft.com/windows2000/techinfo/reskit/default.asp`

A number of third party tools are also available to monitor Windows performance. Sysinternals provides a number of particularly useful tools that may be downloaded at no charge:

`http://www.sysinternals.com`

This section provides a brief description of Windows facilities for monitoring and tuning the performance of the operating system. Table 5-7 lists tools that are commonly used to monitor various resources of interest (this is not a comprehensive list).

Table 5-7 Suggested tool usage

Performance problem	**Suggested tools**
High CPU utilization	Task Manager, Performance Monitor
Not enough memory	Task Manager, Performance Monitor
Paging and/or swapping	Performance Monitor, Memory Visualizer
Disk or filesystem	Task Manager, Performance Monitor, filemon

This section covers the following topics:

- Mapping filesystems to physical disks
- filemon
- Memory Visualizer
- Performance Monitor
- Task Manager
- DB2 Performance Expert for Multiplatforms, described in 2.6.6, “DB2 Performance Expert” on page 88

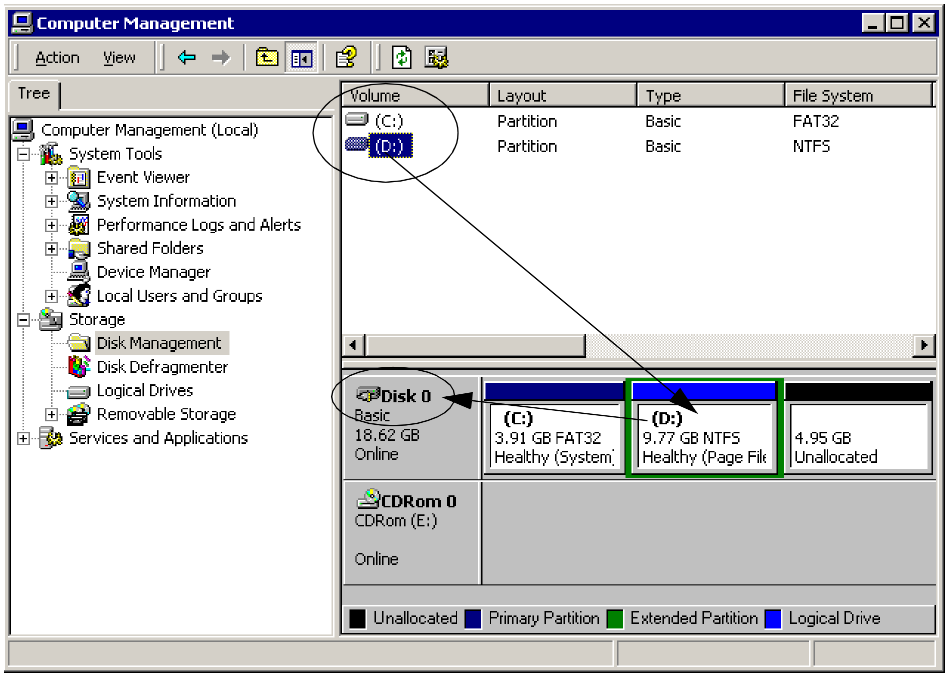
A brief overview of each tool is provided, along with sample output.

Mapping filesystems to physical disks

Windows filesystems are placed on volumes when the volumes are formatted. Volumes are created by partitioning disks and defining volumes in unallocated disk space. Use the Disk Management tool for these tasks. This tool can be found via **Start** -> **Settings** -> **Control Panel** -> **Adminstrative Tools** -> **Computer Management**.

The disk management tool shows the relationship between filesystems (for example, C:\ and D:\) and disks (for example, Disk 0). This is relatively simple for single IDE or SCSI disks.

RAID disk controllers typically present one or more logical disks as, for example, Disk 0, Disk 1, and so on. Identification of the underlying physical disks for each logical disk depends on the RAID device driver, and varies between different device drivers or RAID products.



Volume	Layout	Type	File System
(C:)	Partition	Basic	FAT32
(D:)	Partition	Basic	NTFS

Disk 0
Basic
18.62 GB
Online

(C:)
3.91 GB FAT32
Healthy (System)

(D:)
9.77 GB NTFS
Healthy (Page File

4.95 GB
Unallocated

CDRom 0
CDRom (E:)

Online

Figure 5-27 Windows 2000 Disk Management tool

filemon

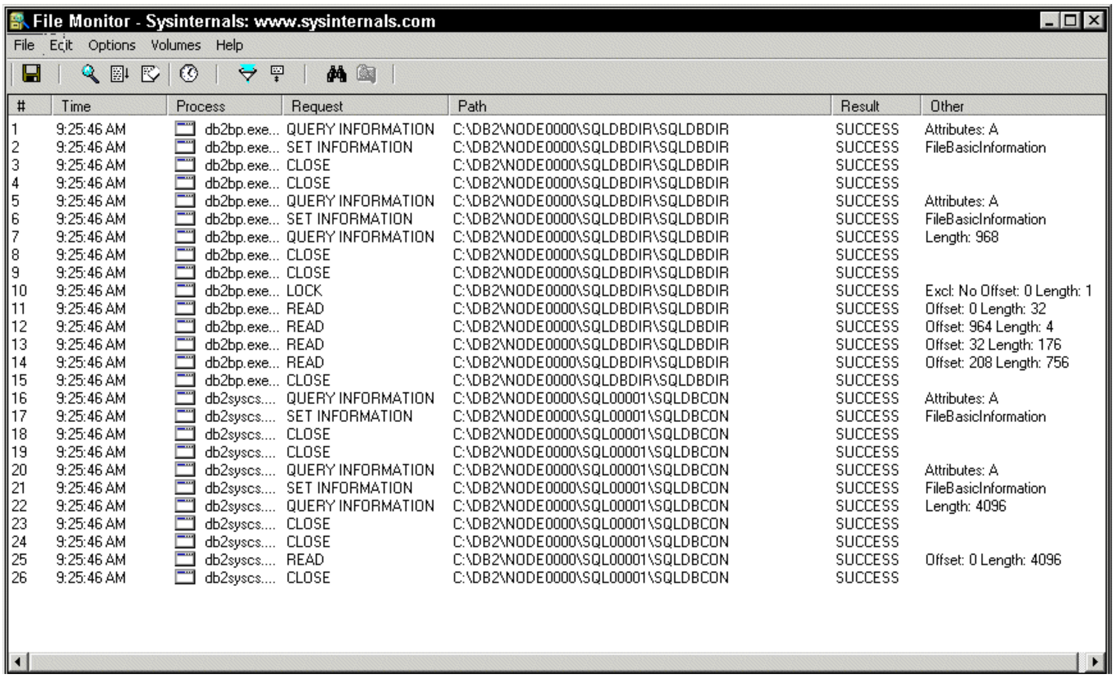
filemon monitors and displays filesystem activity in real time, which makes it a powerful tool for tracking disk and filesystem performance problems. By default, it generates a lot of output since it records every read and write to every file; however, output can be filtered to reduce the volume of data produced.

Note: filemon is a public domain tool from Sysinternals; see:

http://www.sysinternals.com

Figure 5-28 on page 390 shows the output from **filemon** that was filtered to show only files in the DB2 database directory; this was done by selecting **Options** -> **Filter/Highlight** and changing the Include: selection to C:\DB2\*.

This display shows file accesses during a **connect to <database>** statement.



```
#   Time        Process       Request            Path                                   Result    Other
1   9:25:46 AM  db2bp.exe...  QUERY INFORMATION  C:\DB2\NODE0000\SQLDBDIR\SQLDBDIR      SUCCESS   Attributes: A
2   9:25:46 AM  db2bp.exe...  SET INFORMATION    C:\DB2\NODE0000\SQLDBDIR\SQLDBDIR      SUCCESS   FileBasicInformation
3   9:25:46 AM  db2bp.exe...  CLOSE              C:\DB2\NODE0000\SQLDBDIR\SQLDBDIR      SUCCESS
4   9:25:46 AM  db2bp.exe...  CLOSE              C:\DB2\NODE0000\SQLDBDIR\SQLDBDIR      SUCCESS
5   9:25:46 AM  db2bp.exe...  QUERY INFORMATION  C:\DB2\NODE0000\SQLDBDIR\SQLDBDIR      SUCCESS   Attributes: A
6   9:25:46 AM  db2bp.exe...  SET INFORMATION    C:\DB2\NODE0000\SQLDBDIR\SQLDBDIR      SUCCESS   FileBasicInformation
7   9:25:46 AM  db2bp.exe...  QUERY INFORMATION  C:\DB2\NODE0000\SQLDBDIR\SQLDBDIR      SUCCESS   Length: 968
8   9:25:46 AM  db2bp.exe...  CLOSE              C:\DB2\NODE0000\SQLDBDIR\SQLDBDIR      SUCCESS
9   9:25:46 AM  db2bp.exe...  CLOSE              C:\DB2\NODE0000\SQLDBDIR\SQLDBDIR      SUCCESS
10  9:25:46 AM  db2bp.exe...  LOCK               C:\DB2\NODE0000\SQLDBDIR\SQLDBDIR      SUCCESS   Excl: No Offset: 0 Length: 1
11  9:25:46 AM  db2bp.exe...  READ               C:\DB2\NODE0000\SQLDBDIR\SQLDBDIR      SUCCESS   Offset: 0 Length: 32
12  9:25:46 AM  db2bp.exe...  READ               C:\DB2\NODE0000\SQLDBDIR\SQLDBDIR      SUCCESS   Offset: 964 Length: 4
13  9:25:46 AM  db2bp.exe...  READ               C:\DB2\NODE0000\SQLDBDIR\SQLDBDIR      SUCCESS   Offset: 32 Length: 176
14  9:25:46 AM  db2bp.exe...  READ               C:\DB2\NODE0000\SQLDBDIR\SQLDBDIR      SUCCESS   Offset: 208 Length: 756
15  9:25:46 AM  db2bp.exe...  CLOSE              C:\DB2\NODE0000\SQLDBDIR\SQLDBDIR      SUCCESS
16  9:25:46 AM  db2syscs....  QUERY INFORMATION  C:\DB2\NODE0000\SQL00001\SQLDBCON      SUCCESS   Attributes: A
17  9:25:46 AM  db2syscs....  SET INFORMATION    C:\DB2\NODE0000\SQL00001\SQLDBCON      SUCCESS   FileBasicInformation
18  9:25:46 AM  db2syscs....  CLOSE              C:\DB2\NODE0000\SQL00001\SQLDBCON      SUCCESS
19  9:25:46 AM  db2syscs....  CLOSE              C:\DB2\NODE0000\SQL00001\SQLDBCON      SUCCESS
20  9:25:46 AM  db2syscs....  QUERY INFORMATION  C:\DB2\NODE0000\SQL00001\SQLDBCON      SUCCESS   Attributes: A
21  9:25:46 AM  db2syscs....  SET INFORMATION    C:\DB2\NODE0000\SQL00001\SQLDBCON      SUCCESS   FileBasicInformation
22  9:25:46 AM  db2syscs....  QUERY INFORMATION  C:\DB2\NODE0000\SQL00001\SQLDBCON      SUCCESS   Length: 4096
23  9:25:46 AM  db2syscs....  CLOSE              C:\DB2\NODE0000\SQL00001\SQLDBCON      SUCCESS
24  9:25:46 AM  db2syscs....  CLOSE              C:\DB2\NODE0000\SQL00001\SQLDBCON      SUCCESS
25  9:25:46 AM  db2syscs....  READ               C:\DB2\NODE0000\SQL00001\SQLDBCON      SUCCESS   Offset: 0 Length: 4096
26  9:25:46 AM  db2syscs....  CLOSE              C:\DB2\NODE0000\SQL00001\SQLDBCON      SUCCESS
```

Figure 5-28 filemon utility display

Memory Visualizer

As mentioned earlier, Memory Visualizer monitors the memory usage of DB2 UDB Version 8 databases. It is a GUI-based application that runs under Windows but can be used to display data from Windows or AIX-based DB2 databases.

To open the Memory Visualizer in Windows, select **Start** -> **Programs** -> **IBM DB2** -> **Monitoring Tools** -> **Memory Visualizer**. The Memory Visualizer instance selection window opens. Figure 5-17 on page 371 shows a typical display.

Performance Monitor

Performance Monitor provides a graphical display and logging mechanism for a number of Windows 2000 components such as processor, memory, software subsystems and so on. The components are organized as groups of "Performance Objects" of which there are more than 32, and each object has a number of associated counters. For example:

- The "Process" object has counters for %Processor Time and %User TIme, plus various I/O and page fault counters.

- The “DB2” object has approximately 30 associated counters. Figure 5-29 shows how to add a counter showing idle agents on the DB2CTLSV instance to the Performance Monitor graph.

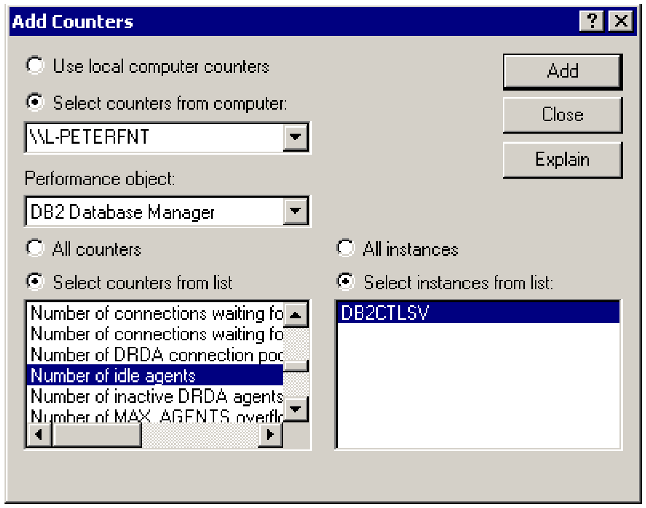
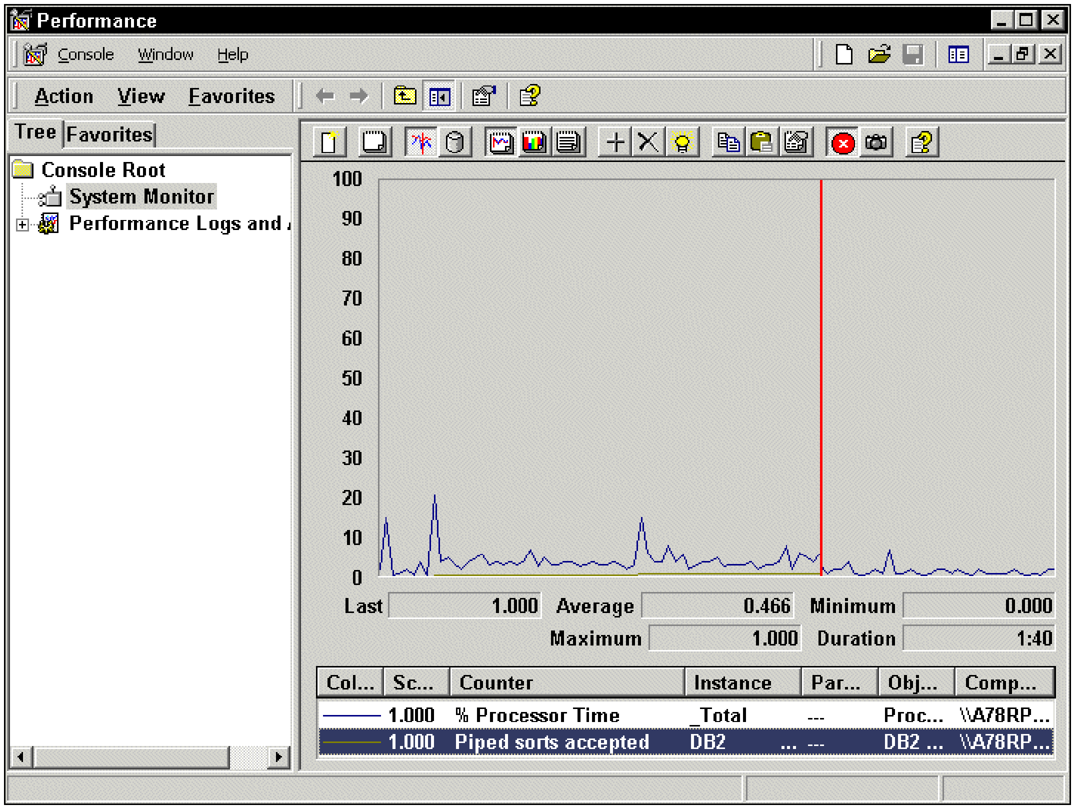

Add Counters
Use local computer counters
Select counters from computer:
\\L-PETERFNT
Performance object:
DB2 Database Manager
All counters
Select counters from list
Number of connections waiting fo
Number of connections waiting fo
Number of DRDA connection poc
Number of idle agents
Number of inactive DRDA agents
Number of MAX_AGENTS overflc
All instances
Select instances from list:
DB2CTLSV
Add
Close
Explain

Figure 5-29 Adding a counter to the Performance Monitor graph

You add counters to the Performance Monitor graph by right-clicking in the graph area. This displays a pop-up which allows you to select “add counters”. Selecting “add counters” displays the screen shown in Figure 5-29. An Explain button provides information about each counter.

Figure 5-30 on page 392 shows processor utilization while running a simple query.



Last 1.000 Average 0.466 Minimum 0.000
Maximum 1.000 Duration 1:40

Col...	Sc...	Counter	Instance	Par...	Obj...	Comp...
——	1.000	% Processor Time	_Total	---	Proc...	\\A78RP...
——	1.000	Piped sorts accepted	DB2	... ---	DB2 ...	\\A78RP...

Figure 5-30 Performance Monitor

Task Manager

The Task Manager can be invoked by Ctrl+Alt+Delete and selecting the **Task Manager** to view the window shown in Figure 5-31 on page 393. The Processes tab provides a large amount of useful information about processes running on the system.

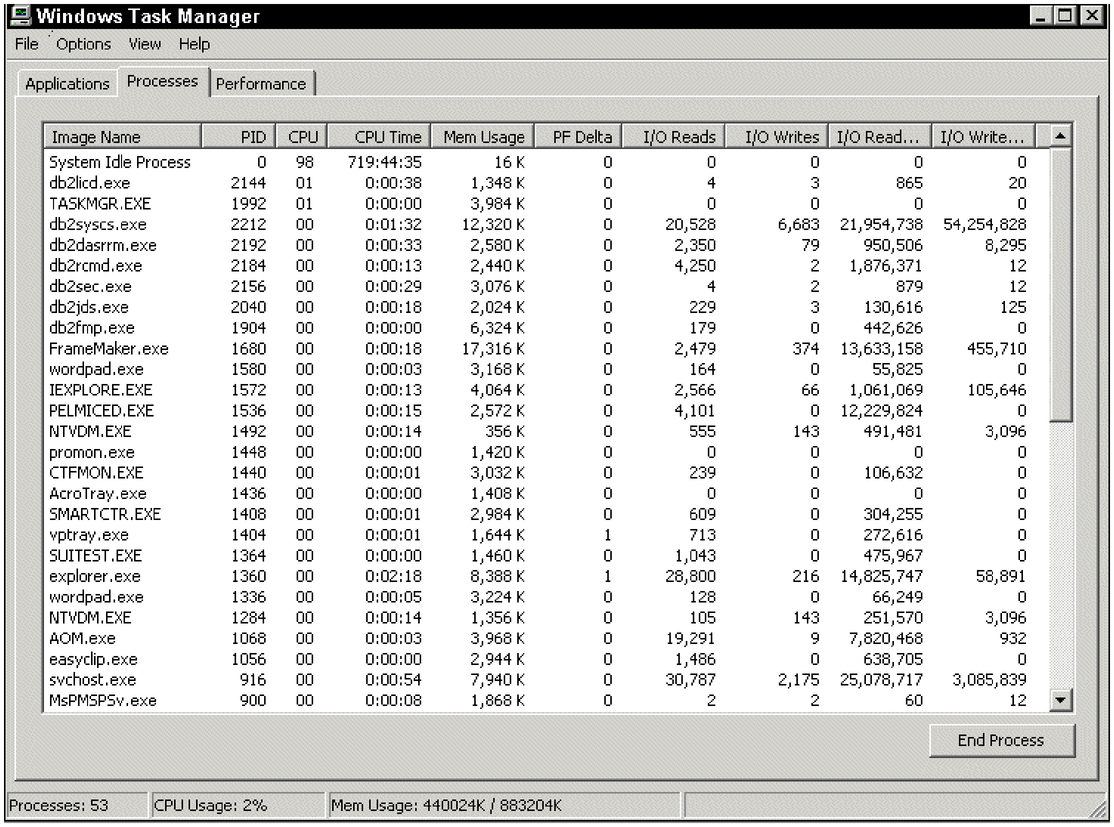


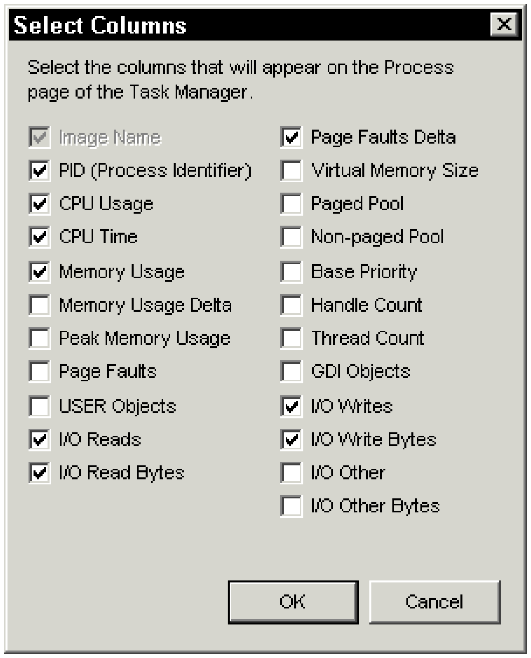
Image Name	PID	CPU	CPU Time	Mem Usage	PF Delta	I/O Reads	I/O Writes	I/O Read...	I/O Write...
System Idle Process	0	98	719:44:35	16 K	0	0	0	0	0
db2licd.exe	2144	01	0:00:38	1,348 K	0	4	3	865	20
TASKMGR.EXE	1992	01	0:00:00	3,984 K	0	0	0	0	0
db2syscs.exe	2212	00	0:01:32	12,320 K	0	20,528	6,683	21,954,738	54,254,828
db2dasrrm.exe	2192	00	0:00:33	2,580 K	0	2,350	79	950,506	8,295
db2rcmd.exe	2184	00	0:00:13	2,440 K	0	4,250	2	1,876,371	12
db2sec.exe	2156	00	0:00:29	3,076 K	0	4	2	879	12
db2jds.exe	2040	00	0:00:18	2,024 K	0	229	3	130,616	125
db2fmp.exe	1904	00	0:00:00	6,324 K	0	179	0	442,626	0
FrameMaker.exe	1680	00	0:00:18	17,316 K	0	2,479	374	13,633,158	455,710
wordpad.exe	1580	00	0:00:03	3,168 K	0	164	0	55,825	0
IEXPLORE.EXE	1572	00	0:00:13	4,064 K	0	2,566	66	1,061,069	105,646
PELMICED.EXE	1536	00	0:00:15	2,572 K	0	4,101	0	12,229,824	0
NTVDM.EXE	1492	00	0:00:14	356 K	0	555	143	491,481	3,096
promon.exe	1448	00	0:00:00	1,420 K	0	0	0	0	0
CTFMON.EXE	1440	00	0:00:01	3,032 K	0	239	0	106,632	0
AcroTray.exe	1436	00	0:00:00	1,408 K	0	0	0	0	0
SMARTCTR.EXE	1408	00	0:00:01	2,984 K	0	609	0	304,255	0
vptray.exe	1404	00	0:00:01	1,644 K	1	713	0	272,616	0
SUITEST.EXE	1364	00	0:00:00	1,460 K	0	1,043	0	475,967	0
explorer.exe	1360	00	0:02:18	8,388 K	1	28,800	216	14,825,747	58,891
wordpad.exe	1336	00	0:00:05	3,224 K	0	128	0	66,249	0
NTVDM.EXE	1284	00	0:00:14	1,356 K	0	105	143	251,570	3,096
AOM.exe	1068	00	0:00:03	3,968 K	0	19,291	9	7,820,468	932
easyclip.exe	1056	00	0:00:00	2,944 K	0	1,486	0	638,705	0
svchost.exe	916	00	0:00:54	7,940 K	0	30,787	2,175	25,078,717	3,085,839
MsPMSPSv.exe	900	00	0:00:08	1,868 K	0	2	2	60	12

Processes: 53   CPU Usage: 2%   Mem Usage: 440024K / 883204K

Figure 5-31 Task Manager process display ranked by CPU utilization

By default, the process list is ranked by CPU utilization. To change the ranking column, simply click that column's heading. To reverse the ranking order, that is, lowest to highest, click the heading again.

The columns to be viewed in this window can be selected through the **View/Select Columns** Taskbar option, as shown in Figure 5-32 on page 394. We suggest adding the following useful columns to your Task Manager: Memory Usage, Memory Usage Delta, I/O Reads, I/O Read Bytes, I/O Writes and I/O Write Bytes.


Select Columns

Select the columns that will appear on the Process page of the Task Manager.

✓ Image Name	✓ Page Faults Delta
✓ PID (Process Identifier)	☐ Virtual Memory Size
✓ CPU Usage	☐ Paged Pool
✓ CPU Time	☐ Non-paged Pool
✓ Memory Usage	☐ Base Priority
☐ Memory Usage Delta	☐ Handle Count
☐ Peak Memory Usage	☐ Thread Count
☐ Page Faults	☐ GDI Objects
☐ USER Objects	✓ I/O Writes
✓ I/O Reads	✓ I/O Write Bytes
✓ I/O Read Bytes	☐ I/O Other
	☐ I/O Other Bytes

Figure 5-32 Task Manager Select Column panel

6

Problem determination scenarios

In this chapter, we discuss some commonly encountered problems in a DB2 OLTP and BI environment, and describe scenarios for identifying and resolving such problems.

The topics covered include:

- DB2 hypotheses hierarchy
- Exception events scenarios
- Online/Realtime event monitoring scenarios
- Routine monitoring scenarios

6.1 Introduction

Most IT environments today are complex infrastructures involving heterogeneous network, hardware, and software components organized in multi-tier configurations. Business applications share this complex IT infrastructure, which is managed by IT professionals skilled in their particular domain of expertise, for example, network administrators, operating system administrators, Web application server administrators, and database administrators.

As mentioned earlier, users may experience performance problems for reasons such as network connectivity and bandwidth constraints, system CPU, I/O and memory constraints, software configuration limitations and constraints, inadequate systems administration skills, poor application design, and faulty assumptions about the workload. In this chapter, we address the following related topics:

1.4, “Problem determination methodology” on page 7 discusses a general problem determination methodology, and recommends a hypotheses validation hierarchy that should typically be followed during problem diagnosis of DB2 applications in general.

For the DB2 application environment shown in Figure 6-1 on page 397, the diagnosis process should sequentially eliminate the cause of the problem as follows:

1. Network-related - between the client and the Web application server
2. Web application server-related - both system (CPU, I/O, memory) and various configuration settings
3. Network-related - between the Web application server and the database server
4. Database server-related - system (CPU, I/O, memory), configuration settings, and routine DBA maintenance activities such as collecting statistics or reorganizing tables
5. Application design-related - tables and SQL

Important: Following this sequence is strongly recommended in order to ensure that the DBA does not expend needless effort on troubleshooting DB2, when the root cause of the performance problem experienced by the user potentially exists elsewhere. For example, network bandwidth constraints or resource contention in the Web application server can manifest as erratic or poor response times for a user of a DB2 application even when the DB2 system and application is perfectly tuned.

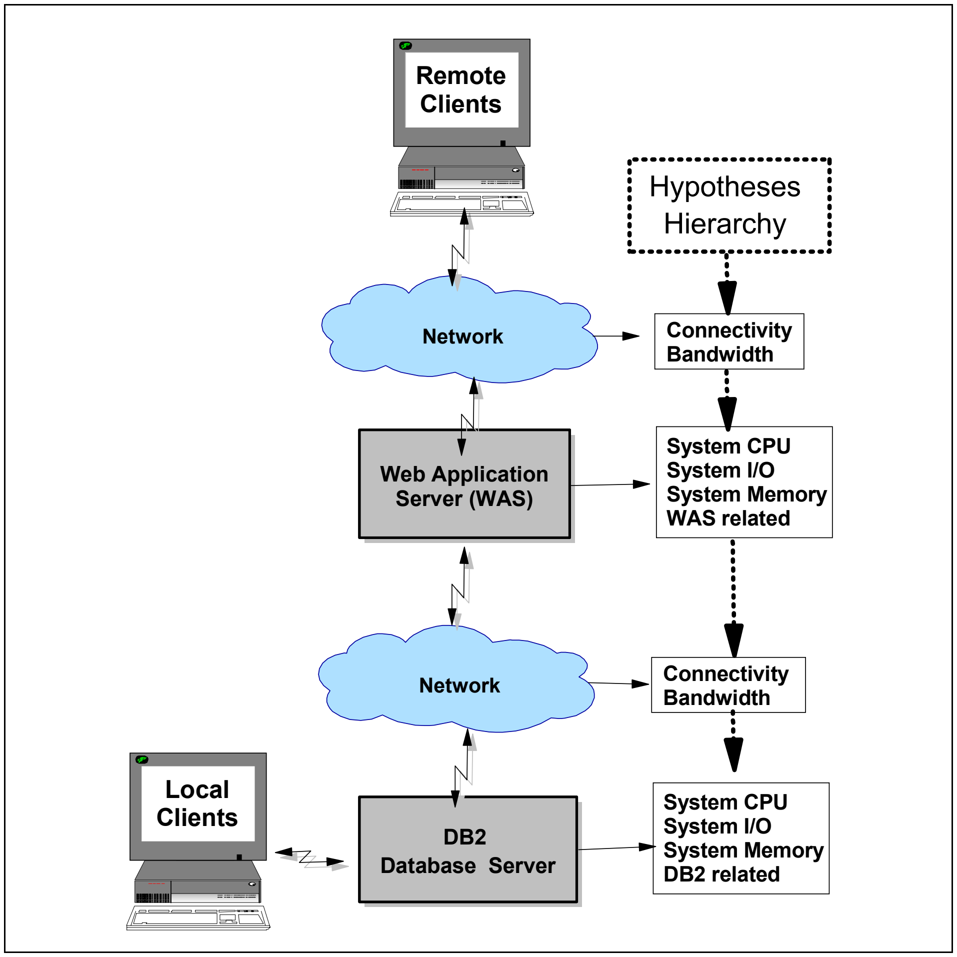


Figure 6-1 A typical DB2 application environment and hypotheses hierarchy

Note: Depending upon the triggering event of the performance problem, it may be possible to skip certain hypotheses validation altogether. For example, when an explicit alert about a lock escalation threshold being tripped is the triggering event, you can ignore hypotheses such as network connectivity and bandwidth constraints, Web application server constraints, and CPU and I/O constraints (in both the Web application server and the DB2 database server) as a potential cause of the problem.

Table 6-1 on page 398 provides a very high level overview of the resource constraint conditions associated with the components that the application utilizes in the execution of its functions, along with tools that can be used to investigate them.

Table 6-1 Typical problem areas associated with DB2 app. performance

Component	Resource constraint conditions	Tools
Network	► Connectivity ► Bandwidth	ping, connect, db2 ping, ftp
System	► CPU utilization ► I/O utilization and performance ► Memory paging	► vmstat, nmon, uptime, ruptime ► iostat, nmon ► vmstat,nmon,svmon,ipcs
Web application server	► System resource constraints ► Connections to DB2 ► Configuration parameters	► as above ► Resource Analyzer ► WebSphere tools
DB2 database server	► System resource constraints ► DB2 resource constraints ► DB2 application design	► as above ► DB2 commands & tools

For a description of AIX tools and examples of their output, see 5.2.4, “Monitoring and problem determination tools” on page 363. For Windows tools, see 5.3.4, “Monitoring and problem determination tools” on page 387. For details on WebSphere Application Server, refer to the IBM Redbook *DB2 UDB/WebSphere Performance Tuning Guide*, SG24-6417.

Important: In most cases, the DBA has no jurisdiction over monitoring and tuning network, system, and Web application server performance drivers, since they are the responsibility of the appropriate administrator.

The objective here is to make DBAs aware of the critical impact on the performance of their DB2 environment by the various network, operating system and Web application server performance drivers, so that they may be in a position to negotiate more effectively with the appropriate administrators responsible for these performance drivers.

We focus on DB2-related resource constraint conditions in the following section, and recommend the hypotheses hierarchy to adopt for problem diagnosis.

Note: We used DB2 UDB Version 8.1.1 in all the scenarios discussed in this section.

6.2 DB2 hypotheses hierarchy

Once the network and systems have been tentatively eliminated as the potential source of the performance problems, we need to narrow the focus to the database server itself.

However, within the database server itself, there is a definite hierarchy of hypotheses validation that must be adopted for effective problem diagnosis, as shown in Figure 6-2.

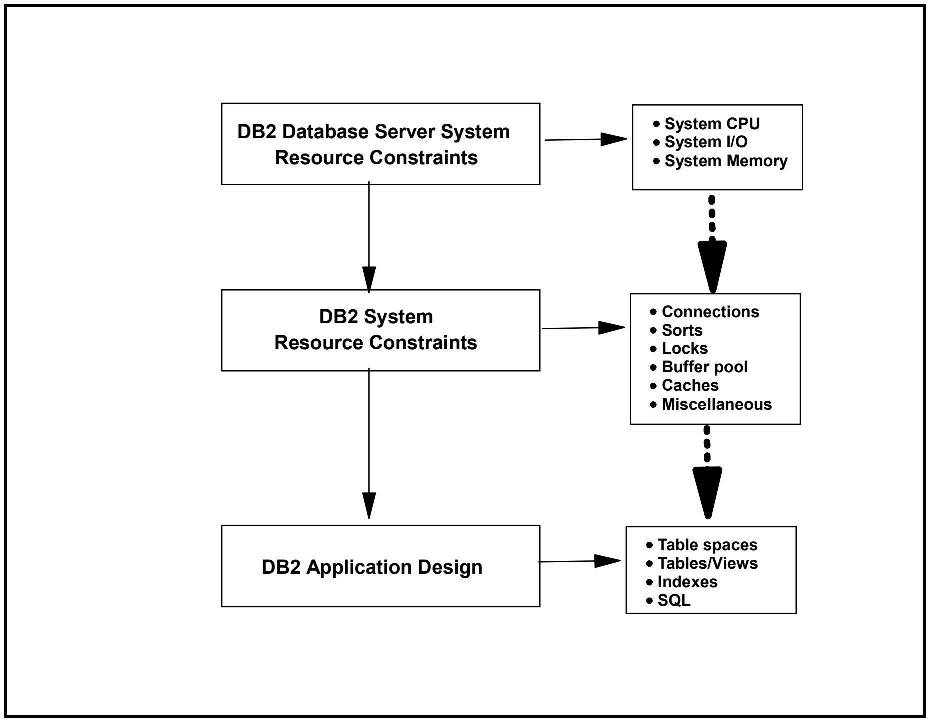


Figure 6-2 DB2 hypotheses hierarchy

The hierarchy is as follows:

1. DB2 database server system resource constraints
2. DB2 system resource constraints
3. DB2 application design-related

Resource constraint conditions associated with each of these items are described in the following subsections.

6.2.1 DB2 database server system resource constraints

The system on which the DB2 database server is running needs to be monitored to ensure that CPU, I/O, and memory consumption is within normal bounds before validating hypotheses further down in the hierarchy.

In some cases, this monitoring of the system resources in general, and DB2 processes in particular, may highlight potential problem areas that need further investigation lower down in the hierarchy to pinpoint the problem; for example, high I/O utilization may indicate excessive contention due to poor placement of highly active tablespaces on the same drive.

Note: Checking for whether DB2 processes are running is a special case of system-level resource consumption checking.

The AIX **`ps -ef | grep db2`** command and the Windows Task Manager can be used to identify DB2 processes.

6.2.2 DB2 system resource constraints

Certain DB2 database manager and database configuration settings may result in applications suffering response time problems due to the following:

- Connection constraints
- Sorting constraints
- Locking constraints
- Buffer pool constraints
- Cache size constraints, such as catalog and package
- Miscellaneous constraints, such as enabling intra-partition parallelism

Important: These constraints are *not* ordered by the impact they have on performance, but by the sequence in which problem diagnosis is recommended.

In many cases, problem diagnosis is a holistic affair where a particular problem is best diagnosed by combining monitoring results of multiple events and entities.

Important: This section does *not* discuss every monitoring and tuning knob available in DB2 to manage the performance of the DB2 environment. The purpose of this section is to promote a top-down discipline to the process of problem diagnosis by discussing some of the more important DB2 resource constraints that impact DB2 performance.

In practice, the impact of a particular resource constraint on overall DB2 performance depends a great deal upon the nature and priorities of the application workload and the resources available for its execution. For example, with a read-only data warehousing type of workload, locking constraints are likely to play an insignificant role in overall DB2 performance, while sorting constraints will probably have considerable performance impact.

The following subsections identify many of the key database manager configuration and database configuration parameters associated with each of these constraints, and describe some of the monitoring elements that should be used to configure them optimally.

Note: Database manager configuration settings can be viewed by issuing the following command:

```
db2 get dbm cfg
```

Database configuration settings can be viewed by issuing the following command:

```
db2 get db cfg for <database_name>
```

The monitoring elements are reported in the snapshot monitor, and may represent water marks, counters, time and gauges.

Attention: When the monitoring elements are *not* high water marks, they should be sampled at specific intervals over an extended period of time to get a realistic view of system usage.

In some cases, it may be necessary to reset counters at the start of the monitoring interval to get an accurate view of activity.

Connection constraints

Connection problems are generally manifested as messages generated by an application when a connection exception is returned to the application. The following database manager configuration and database configuration parameters can constrain the number of connections permitted:

- Database manager configuration parameters
  - max_connections
  - maxagents
  - maxcagents
  - max_coordagents
- Database configuration parameters
  - maxappls

To determine whether connections problems are being experienced, the snapshot monitor should be invoked during the appropriate monitor interval, and relevant fields checked, as follows:

- For the database manager configuration parameters **`max_connections, maxagents, maxcagents`** and **`max_coordagents`**, relevant snapshot contents are shown in Example 6-1.

Example 6-1 dbm snapshot for connections

```
db2 => get snapshot for dbm
...
Remote connections to db manager             = 0
Remote connections executing in db manager   = 0
Local connections                            = 4
Local connections executing in db manager    = 0
Active local databases                       = 1

High water mark for agents registered        = 5
High water mark for agents waiting for a token = 0
Agents registered                            = 5
Agents waiting for a token                   = 0
Idle agents                                  = 0
....
Agents assigned from pool                    = 2
Agents created from empty pool               = 7
Agents stolen from another application       = 0
High water mark for coordinating agents      = 5
Max agents overflow                          = 0
Hash joins after heap threshold exceeded     = 0
.....
```

Check the following values:

- Sum of (**Remote connections to db manager** + **Local connections**) should be less than **`max_connections`**.

Note: These values are *not* high water marks, and should therefore be sampled at specific intervals over an extended period of time and at representative intervals to get a realistic view of system usage.

If this value is the same as the **max_connections**, then it is likely that some database connection requests were rejected.

- **High water mark for agents registered** should be less than **maxagents**.

  If this value is the same as the **maxagents**, then it means that connections may have failed.

- **High water mark for agents waiting for a token** should be equal to zero in unconstrained systems.

  If this value is the same as the **maxcagents**, then it means that certain applications had to wait, since a transaction cannot be initiated without getting a token.

- **High water mark for coordinating agents** should be less than **max_coordagents**.

  If this value is the same as the **max_coordagents**, then it means that agent creations may have failed.

- For the database configuration parameters **maxappls**, relevant snapshot contents are shown in Example 6-2.

Example 6-2 db snapshot for connections

```
db2 => get snapshot for all on sample
...
High water mark for connections            = 3
Application connects                       = 3
Secondary connects total                   = 0
Applications connected currently           = 3
Appls. executing in db manager currently   = 0
Agents associated with applications        = 3
Maximum agents associated with applications= 3
Maximum coordinating agents                = 3
...
```

Check that the **High water mark for connections** is less than **maxappls**, unless **maxappls** is specified as being AUTOMATIC.

If this value is the same as the **maxappls** parameter, then it is likely that some database connection requests were rejected.

Sorting constraints

Sorting problems are generally manifested as poor response times for the application, and as the occasional application error message when sort heap hard limits are exceeded.

The following database manager configuration and database configuration parameters can impact sort performance:

- Database manager configuration parameters
  - **`sheapthres`**
- Database configuration parameters
  - **`sheapthres_shr`**
  - **`sortheap`**

To determine whether sorting problems are being experienced, the snapshot monitor should be invoked during the appropriate monitor interval, and relevant fields checked, as follows:

- For the database manager configuration parameter **`sheapthres`**, relevant snapshot contents are shown in Example 6-3.

Example 6-3 dbm snapshot for sorting

```
db2 => get snapshot for dbm
...
Private Sort heap allocated                 = 0
Private Sort heap high water mark           = 277
Post threshold sorts                        = 0
Piped sorts requested                       = 10
Piped sorts accepted                        = 10
...
Agents assigned from pool                   = 2
Agents created from empty pool              = 7
Agents stolen from another application      = 0
High water mark for coordinating agents     = 5
Max agents overflow                         = 0
Hash joins after heap threshold exceeded    = 0
...
```

  Check the following values:

  - **Private Sort heap high water mark** should be less than **`sheapthres`**.

    If this value is greater than or equal to **`sheapthres`**, then it means that the sorts are not getting the full sort heap as specified by the **`sortheap`** database configuration parameter.
  - **Post threshold sorts** should be very small, if not zero.

When post threshold sorts occur, the database manager allocates a smaller sort heap than that specified by the **`sortheap`** database configuration parameter. Subsequent sort heap allocations are reduced even further until the total amount of sort heap in use falls below the amount specified for **`sheapthres`**. This situation causes a serious degradation in database performance and should be avoided.

> **Note:** This value is *not* a high water mark, and should therefore be sampled at specific intervals over an extended period of time and at representative intervals to get a realistic view of such occurrences.

– Difference between **Piped sorts requested** and **Piped sorts accepted** should be very small, if not zero.

  If this value is high, then it means that piped sorts are being rejected because the sort heap threshold (**`sheapthres`**) would be exceeded when the sort heap is allocated for the sort.

> **Note:** Here too, these values are *not* a high water marks, and should therefore be sampled at specific intervals over an extended period of time and at representative intervals to get a realistic view of such occurrences.

– **Hash joins after heap threshold exceeded** should be very small, if not zero.

  If this value is non-zero, then it means that it means that a hash join heap request was limited because of the sort heap threshold (**`sheapthres`**) being exceeded. This value should be used in conjunction with **Hash join overflows** to calculate the percentage of such occurrences.

> **Note:** This value is *not* a high water mark, and should therefore be sampled at specific intervals over an extended period of time and at representative intervals to get a realistic view of such occurrences.

- For the database configuration parameters **`sheapthres_shr`** and **`sortheap`**, relevant snapshot contents are shown in Example 6-4.

Example 6-4 db snapshot for sorting

```
db2 => get snapshot for all  on sample
...
Total Private Sort heap allocated          = 0
Total Shared Sort heap allocated           = 0
Shared Sort heap high water mark           = 0
```

```
Total sorts                                  = 1
Total sort time (ms)                         = Not Collected
Sort overflows                               = 1
Active sorts                                 = 0
...
```

Check the following values:

– **Shared Sort heap high water mark** should be less than `sheapthres_shr`.

  If this value is equal to `sheapthres_shr`, then it means that the total amount of database shared memory that can be used for sorting at any one time has been exceeded, and subsequent sorts will fail with an `SQL0955C` message.

– **Sort overflows** should be very small, if not zero.

  A non-zero value indicates that sorts ran out of sort heap (`sortheap`) and had to overflow to disk. This value should be used in conjunction with **Total sorts** to calculate the percentage of sorts that overflowed to disk.

  Note: This value is *not* a high water mark, and should therefore be sampled at specific intervals over an extended period of time and at representative intervals to get a realistic view of such occurrences.

Locking constraints

Locking problems are generally manifested as poor response times, and the occurrences of deadlocks and timeouts. Certain locking problems, such as lock escalations and deadlocks, are also reported in the db2diag.log.

The following database configuration parameters can severely impact the amount of concurrency and throughput achievable.

- Database configuration parameters
  – dlchktime
  – locklist
  – locktimeout
  – maxlocks

To determine whether locking problems are being experienced, the snapshot monitor should be invoked during the appropriate monitor interval, and relevant fields should be checked as shown in Example 6-5.

Example 6-5 db snapshot for locking

```
db2 => get snapshot for all  on sample
...
Locks held currently                         = 1
```

```
Lock waits                                 = 0
Time database waited on locks (ms)         = 0
Lock list memory in use (Bytes)            = 540
Deadlocks detected                         = 0
Lock escalations                           = 0
Exclusive lock escalations                 = 0
Agents currently waiting on locks          = 0
Lock Timeouts                              = 0
...
```

Check the following values:

- The ratio of **Time waited on locks** and **Lock waits** is a measure of the average duration of each lock wait, and should be subsecond or a few seconds only.

  If this number is high, it could be due to applications holding too many locks, for extended durations, or lock escalations. It is also possible that extended lock wait durations may point to a problem with the **`locktimeout`** value.

- **Lock escalations** and **Exclusive lock escalations** should be kept to a minimum.

  If a large number of escalations are seen over a short interval, then it could mean that the **`locklist`** and/or **`maxlocks`** values are too small. These values should be evaluated in conjunction with **Deadlocks detected**, **Lock waits**, and **Time waited on locks.**

  Note: Further detailed information about lock escalations can be obtained from the *d*b2diag.log.

- **Deadlocks detected** and **Lock timeouts** should be kept to a minimum.

  If a large number of deadlocks and lock timeouts are seen over a short interval, then it could mean that the **`dlchktime`** value is too large and the **`locktimeout`** value is too small. Here again, these values should be evaluated in conjunction with **Lock escalations, Exclusive lock escalations, Lock waits**, and **Time waited on locks.**

Note: In all these cases, the monitoring elements are counters or gauges representing a specific point in time, and should therefore be sampled at peak or heavy workload intervals to ascertain whether locking problems exist; this should involve resetting the counters at the start of the monitoring interval.

Buffer pool constraints

Buffer pool problems are generally manifested as poor response times as a result of increased synchronous I/Os or operating system paging. The key metric to measure here is the hit ratio achieved (the higher, the better).

Buffer pools are defined via the CREATE BUFFERPOOL SQL statement, and are allocated at database activation. Buffer pools are used by tables, indexes, and sort temporary tablespaces.

Note: Buffer pool information is typically gathered at a tablespace level, but the facilities of the Database System Monitor can roll this information up to the buffer pool and database levels.

To determine whether buffer pool problems are being experienced, the Snapshot Monitor should be invoked during the appropriate monitor interval, and relevant fields checked as shown in Example 6-6.

Example 6-6 db snapshot for buffer pools

```
$ db2 "get snapshot for all on dtw"
.....
Buffer pool data logical reads             = 5160
Buffer pool data physical reads            = 1485
Buffer pool data writes                    = 0
Buffer pool index logical reads            = 18371
Buffer pool index physical reads           = 3064
Total buffer pool read time (ms)           = 69774
Total buffer pool write time (ms)          = 0
Asynchronous pool data page reads          = 0
Asynchronous pool data page writes         = 0
Buffer pool index writes                   = 0
Asynchronous pool index page reads         = 16
Asynchronous pool index page writes        = 0
Total elapsed asynchronous read time       = 268
Total elapsed asynchronous write time      = 0
Asynchronous read requests                 = 0
Direct reads                               = 236
Direct writes                              = 0
Direct read requests                       = 58
Direct write requests                      = 0
Direct reads elapsed time (ms)             = 483
Direct write elapsed time (ms)             = 0
Database files closed                      = 0
Data pages copied to extended storage      = 0
Index pages copied to extended storage     = 0
Data pages copied from extended storage    = 0
Index pages copied from extended storage   = 0
```

```
Unread prefetch pages                    = 0
Vectored IOs                             = 0
Pages from vectored IOs                  = 0
Block IOs                                = 0
Pages from block IOs                     = 0
Physical page maps                       = 0
```

- Compute the total buffer pool hit ratio as:

  (1 - ((**Buffer pool data physical reads** + **Buffer pool index physical reads**) / (**Buffer pool data logical reads** + **Buffer pool index logical reads**))) * 100%

- Compute the index hit ratio as:

  (1 - ((**Buffer pool index physical reads**) / (**Buffer pool index logical reads**))) * 100%

Note: In this case, the monitoring elements are counters representing a specific point in time, and should therefore be sampled at peak or heavy workload intervals to ascertain whether buffer pool problems exist; this should involve resetting the counters at the start of the monitoring interval.

Cache size constraints

There are several caches in DB2, and problems with their size are generally manifested as poor response times as a result of a increased synchronous I/Os or operating system paging.

Note: There are several database manager and database caches that can be specified such as **aslheapsz, audit_buf_sz, java_heap_sz, query_heap_sz,** and **applheapsz**, to name a few.

The objective is to determine if these caches are *too small* or *too big*. Each of these has slightly different monitor elements to look at, as described in Chapter 3, “Application design and system performance considerations” on page 107.

We focus here on **catalogcache_sz** and **pckcachesz**. The key metrics to assess the efficiency of these caches are hit ratios, overflows, and high water marks.

To determine whether **catalogcache_sz** and **pckcachesz** problems are being experienced, the snapshot monitor should be invoked during the appropriate monitor interval, and relevant fields checked, as shown in Example 6-7 on page 410.

Example 6-7 db snapshot for catalogcache_sz and pckcachesz

```
db2 => get snapshot for all  on sample
...
Package cache lookups                      = 4
Package cache inserts                      = 2
Package cache overflows                    = 0
Package cache high water mark (Bytes)      = 118968
Application section lookups                = 6
Application section inserts                = 2

Catalog cache lookups                      = 13
Catalog cache inserts                      = 5
Catalog cache overflows                    = 0
Catalog cache high water mark              = 0
...
```

- The hit ratio for the package cache is computed as follows:

  `(1 - (Package cache inserts / Package cache lookups)) * 100%`

- **Package cache overflows** should be kept to zero.

  If this number is non-zero, then it means a spillover has occurred into other heaps, such as locklist, thereby resulting in unnecessary lock escalation and general performance degradation.

- **Package cache high water mark (Bytes)** represents the largest size reached by the package cache, and can be used to determine the value for the **`pckcachesz`** database configuration parameter to avoid cache overflows.

  Note: Except for the **Package cache high water mark (Bytes)** monitoring element, the other elements reported are counters representing a specific point in time, and should therefore be sampled at peak or heavy workload intervals to ascertain whether package cache size problems exist; this should involve resetting the counters at the start of the monitoring interval.

  Important: The same considerations as discussed for **`pckcachesz`** also apply to **`catalogcache_sz`**.

Miscellaneous constraints

There are a host of other database manager and database configuration parameters that can impact the performance of DB2 applications. Details of their individual performance impact, and monitoring and tuning considerations, are described in Chapter 3, “Application design and system performance considerations” on page 107.

Some of the parameters of interest are as follows:

- Database manager configuration parameters
  - intra_parallel
  - maxfilop and maxtotfilop
  - rqrioblk
  - num_poolagents
- Database configuration parameters
  - chngpgs_thresh
  - logbufsz
  - num_iocleaners
  - num_ioservers
- Registry and environment variables
  - DB2_PARALLEL_IO
  - DB2MEMDISCLAIM
  - DB2MEMMAXFREE
  - DB2NTMEMSIZE
  - DB2_PINNED_BP

DB2 application design-related

The monitoring of DB2 system resource constraints on page 400, and DB2 resource constraints on page 400, may either fail to identify any cause of performance problems, or provide clues to potential performance problems in the application such as SQL access paths and faulty design.

- An example of SQL access path problems might be a missing index resulting in a table scan, while a faulty design example might be an inappropriate choice of dimension keys for an MDC table.
- An example of a clue in DB2 resource constraints that points to an application design-related SQL access path problem of a missing index, might be a discovery of excessive sort times for an application in an OLTP environment.

Attention: Pinpointing a specific poorly performing application and SQL statement *may* require the DBA to initiate exception monitoring activities such as activating monitor switches and issuing relevant **`get snapshot`** commands, raising the diaglevel, and creating and activating Event Monitors under controlled conditions (that is, for short bursts during the problem interval).

The performance of an SQL statement depends upon many factors including:

- Poor access path selection due to missing indexes, missing or outdated statistics, disorganized tables, or data distribution profile changes.
- Poor application design such as poorly written SQL, or poorly designed tables.

Poor access path selection

The process of verifying access path selection involves the following:

- Using EXPLAIN
- Identifying available indexes
- Ensuring tables/indexes are well organized and do not require reorganization
- Ensuring statistics are up to date
- Using Design Advisor for index recommendations

These tasks can be accomplished via the Control Center, or by using DB2 commands via the Command Line Processor.

Poor application design

This requires detailed analysis of the application in question, and applying best practices as described in 3.3, "Application design considerations" on page 111.

6.3 Exception event scenarios

Exception events frequently manifest themselves in the form of user complaints about poor response times, or unexpected application error messages including timeouts and deadlocks. Adopting a consistent problem determination methodology as described in "Problem determination methodology" on page 7 will ensure speedy resolution of performance problems.

The following subsections cover the following problem determination scenarios:

1. Lock waits due to default LOCKTIMEOUT value (OLTP)
2. Poor SQL performance due to missing indexes (OLTP)
3. Poor SQL performance due to unused MQTs (BI)

Note the following points in regard to the scenarios:

1. The workload and environments were artificially contrived to produce the relevant problem condition for the problem diagnosis exercise; therefore, certain settings can clearly been seen to be "inappropriate" in real world environments.
2. The emphasis of these scenarios is on problem diagnosis, and *not* on problem resolution per se. Best practices for problem *resolution* are

discussed briefly, but not applied to demonstrate the elimination of the problem.

We have organized the description of each scenario as follows:

- Description of the application
- Environment configuration
- Monitor level settings
- Workload used
- Triggering event
- Hypotheses and validations
- Root cause of the problem
- Apply best practices

6.3.1 Lock waits due to default LOCKTIMEOUT value (OLTP)

Lock waits are a common and natural occurrence in any multi-user application environment. A well-designed application will minimize lock waits by using appropriate locking levels when accessing shared data. However, from time to time, excessive numbers of lock waits or lock waits for extended periods may give rise to user complaints about poor performance.

In this scenario, we show how a lock wait problem caused by the default value of LOCKTIMEOUT causes a performance problem. The default LOCKTIMEOUT value of -1 will cause one or more applications to wait indefinitely for rows or tables that are locked by another application.

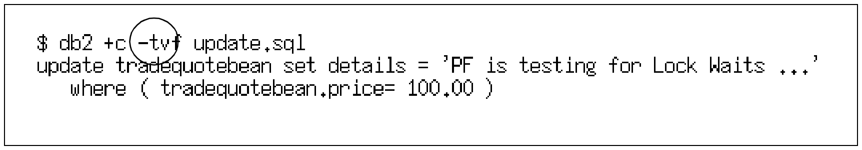
System defaults are often accepted in the real world and can lead to the kinds of problems described here.

Description of the application

We used the Trade2 OLTP application, and some additional uncommitted SQL statements simulating contending applications, to demonstrate this scenario. The Trade2 application is described in Appendix B, “Workloads used in the scenarios” on page 485.

The SQL statements used in the application are described in Figure 6-3 on page 414 and Figure 6-4 on page 414. Figure 6-3 shows a batch file consisting of an SQL UPDATE statement that does not commit. This will result in all updated rows being locked until a commit is issued.

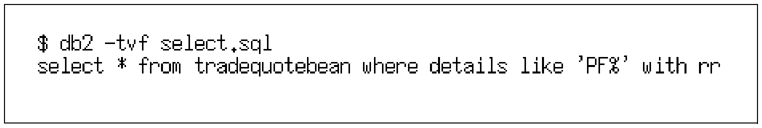
Note: Our intention with the batch file in Figure 6-3 was to simulate the problem of applications performing “infrequent commits”, which can undermine concurrency and result in erratic response times.



```
$ db2 +c -tvf update.sql
update tradequotebean set details = 'PF is testing for Lock Waits ...'
   where ( tradequotebean.price= 100.00 )
```

Figure 6-3 lock timeout example update statement

Figure 6-4 shows a second batch file that consists of an SQL SELECT statement that attempts to read the rows being updated by the batch file shown in Figure 6-3. Since the requested rows are locked, the application represented by the batch file shown in Figure 6-4 has to wait until the lock is released by the batch file application shown in Figure 6-3.



```
$ db2 -tvf select.sql
select * from tradequotebean where details like 'PF%' with rr
```

Figure 6-4 lock timeout example select statement

Environment configuration

We used a single AIX pSeries Model 270 to run the WebSphere Application Server and DB2 database, and a Windows 2000 PC to drive the Trade2 application on the WebSphere Application Server. The uncommitted SQL applications accessed DB2 from a Windows 2000 UNIX X terminal emulator.

Table 6-2 and Figure 6-5 on page 415 highlight the details of the environment.

Table 6-2 LOCKTIMEOUT scenario configuration

Hardware configuration	**Software configuration**
pSeries Model 270 2 CPUs 1 GB RAM 2 x 36GB SCSI disks	AIX 5.1 ML03 DB2 V8.1 FP1 WebSphere Application Server 4.0.5
PC	Windows 2000 akstress tool
Laptop	Windows 2000 Exceed X terminal emulator

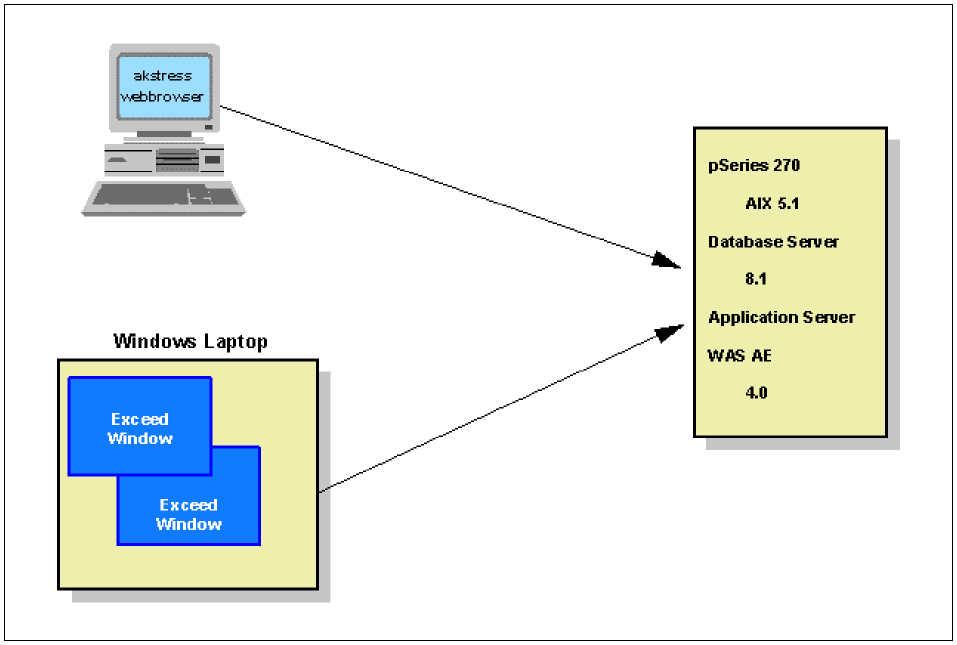
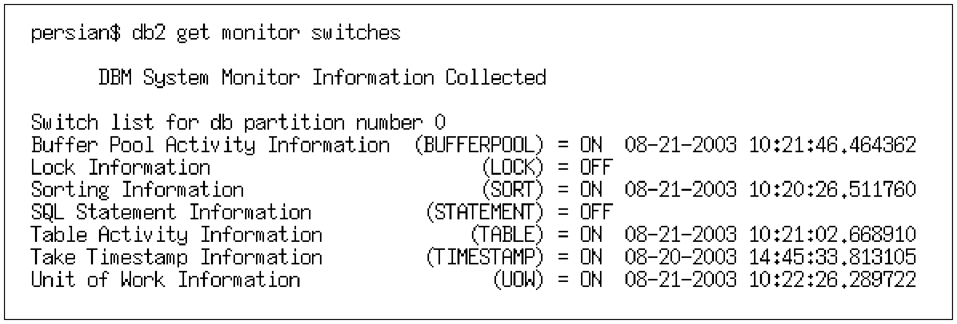


Figure 6-5 LOCKTIMEOUT scenario architecture

Monitor level settings

We used the following recommended settings for routine monitoring in an OLTP environment, as shown in Figure 6-6:

- Set monitor switches **`DFT_MON_BUFPOOL`**, **`DFT_MON_SORT`**, **`DFT_MON_TABLE`**, **`DFT_MON_TIMESTAMP`**, **`DFT_MON_UOW`**, and **`HEALTH_MON`** to ON.
- Set monitor switches **`DFT_MON_LOCK`** and **`DFT_MON_STMT`** to OFF.
- Accept the **`DIAGLEVEL`** default setting (3).



```
persian$ db2 get monitor switches

      DBM System Monitor Information Collected

Switch list for db partition number 0
Buffer Pool Activity Information  (BUFFERPOOL) = ON  08-21-2003 10:21:46.464362
Lock Information                        (LOCK) = OFF
Sorting Information                     (SORT) = ON  08-21-2003 10:20:26.511760
SQL Statement Information          (STATEMENT) = OFF
Table Activity Information             (TABLE) = ON  08-21-2003 10:21:02.668910
Take Timestamp Information         (TIMESTAMP) = ON  08-20-2003 14:45:33.813105
Unit of Work Information                 (UOW) = ON  08-21-2003 10:22:26.289722
```

Figure 6-6 Recommended monitor switch settings for OLTP routine monitoring

Workload used

We used the akstress tool (described in B.4, “Trade2 database and application” on page 487) to simulate 20 users of the Trade2 application, with each user performing a login followed by a single query and logout. We ran this workload for 20 minutes.

The success or failure of each transaction is displayed on an **`akstress`** monitoring window. We assumed, for the purposes of this scenario, that if a significant number of transactions failed for any reason, a user would register a complaint. We also ran the batch files from multiple DB2 command line processor windows.

Triggering event

Several user complaints about not getting any response from the system, while others complained about experiencing erratic response times.

Hypotheses and validations

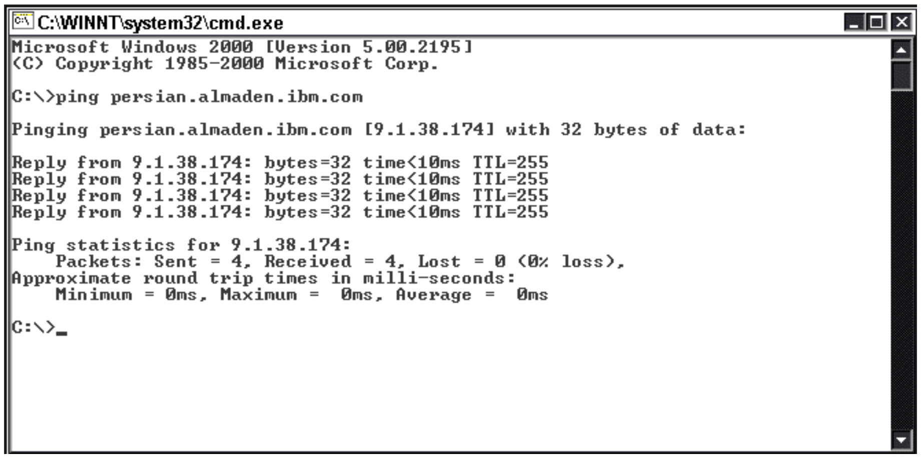
We adopted the hypotheses hierarchy as described in “Problem determination methodology” on page 7, and validated each hypothesis in turn as follows:

- Hypothesis 1: Network performance problems
- Hypothesis 2: WebSphere Application Server performance problems
- Hypothesis 3: DB2 database server system problems
- Hypothesis 4: DB2 system problems
- Hypothesis 5: DB2 application problems

Hypothesis 1: Network performance problems

The network administrator was consulted, and we were advised that the network had been and was operating within normal bounds.

Our own attempts to ping the database server returned very low round trip times, as shown in Figure 6-7 on page 417.



```
Microsoft Windows 2000 [Version 5.00.2195]
(C) Copyright 1985-2000 Microsoft Corp.

C:\>ping persian.almaden.ibm.com

Pinging persian.almaden.ibm.com [9.1.38.174] with 32 bytes of data:

Reply from 9.1.38.174: bytes=32 time<10ms TTL=255
Reply from 9.1.38.174: bytes=32 time<10ms TTL=255
Reply from 9.1.38.174: bytes=32 time<10ms TTL=255
Reply from 9.1.38.174: bytes=32 time<10ms TTL=255

Ping statistics for 9.1.38.174:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum =  0ms, Average =  0ms

C:\>_
```

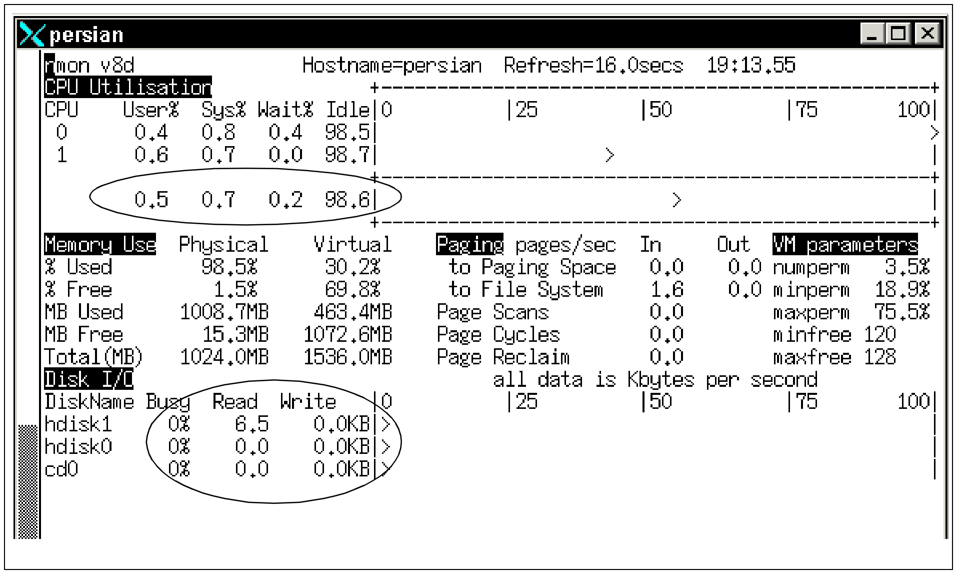
Figure 6-7 ping persian

Both PCs and the server were on the same network. We therefore concluded that network availability and bandwidth was most likely *not* the cause of the performance problem.

Hypothesis 2: WebSphere Application Server problems

In checking for problems with the WebSphere Application Server, we need to check both the system resource consumption of the WebSphere Application Server machine, and the WebSphere Application Server environment.

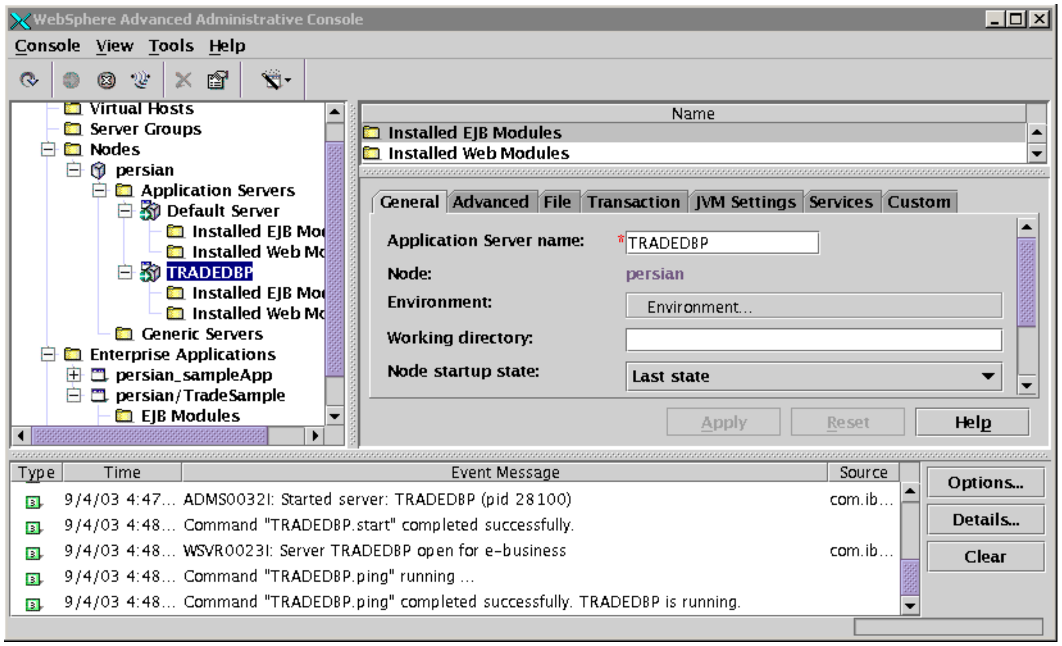
Figure 6-8 on page 418 shows the output of the **nmon** command. The system running the WebSphere Application Server was not busy, and there were no significant IO performance problems. Both the CPU utilization and disk throughput were low.



```
nmon v8d               Hostname=persian  Refresh=16.0secs  19:13.55
CPU Utilisation      +-------------------------------------------------+
CPU    User%  Sys% Wait% Idle|0          |25         |50          |75        100|
 0       0.4   0.8   0.4  98.5|                                                 >
 1       0.6   0.7   0.0  98.7|                    >                            |
                             +-------------------------------------------------+
         0.5   0.7   0.2  98.6|                          >                      |
                             +-------------------------------------------------+
Memory Use  Physical    Virtual    Paging pages/sec   In     Out  VM parameters
% Used        98.5%       30.2%     to Paging Space   0.0    0.0 numperm   3.5%
% Free         1.5%       69.8%     to File System    1.6    0.0 minperm  18.9%
MB Used     1008.7MB    463.4MB    Page Scans         0.0        maxperm  75.5%
MB Free       15.3MB   1072.6MB    Page Cycles        0.0        minfree 120
Total(MB)   1024.0MB   1536.0MB    Page Reclaim       0.0        maxfree 128
Disk I/O                                all data is Kbytes per second
DiskName Busy   Read  Write      |0          |25         |50          |75        100|
hdisk1     0%    6.5    0.0KB|>                                                 |
hdisk0     0%    0.0    0.0KB|>                                                 |
cd0        0%    0.0    0.0KB|>                                                 |
```

Figure 6-8   nmon output

We then checked the WebSphere Application Server console for error messages, as shown in Figure 6-9 on page 419. There were no errors that pointed to any performance problems.



Application Server name: TRADEDBP
Node: persian
Environment: Environment...
Working directory:
Node startup state: Last state

```
Type  Time           Event Message                                                       Source
      9/4/03 4:47... ADMS0032I: Started server: TRADEDBP (pid 28100)                     com.ib...
      9/4/03 4:48... Command "TRADEDBP.start" completed successfully.
      9/4/03 4:48... WSVR0023I: Server TRADEDBP open for e-business                      com.ib...
      9/4/03 4:48... Command "TRADEDBP.ping" running ...
      9/4/03 4:48... Command "TRADEDBP.ping" completed successfully. TRADEDBP is running.
```

Figure 6-9 WebSphere Application Server console

Hypothesis 3: DB2 database server system problems

In our scenario, both WebSphere Application Server and DB2 server shared the same machine. Since Figure 6-8 on page 418 shows the system operating normally, we eliminated this hypothesis as the cause of the performance problem.

Hypothesis 4: DB2 system problems

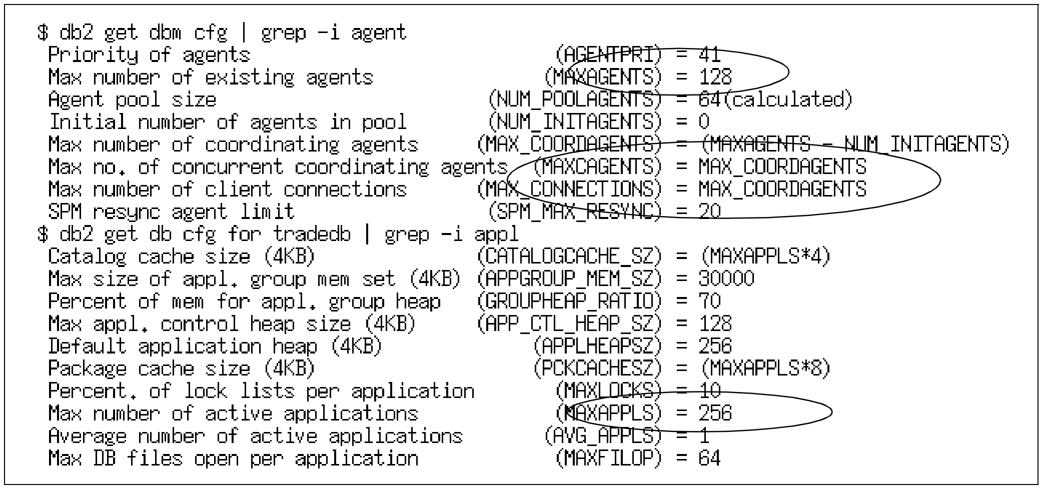
We adopted the following hypotheses hierarchy described in “DB2 system resource constraints” on page 400 to validate the cause of the performance problem:

- Connection problems
- Sorting problems
- Locking problems
- Buffer pool problems
- Cache size problems
- Miscellaneous problems

Connection problems

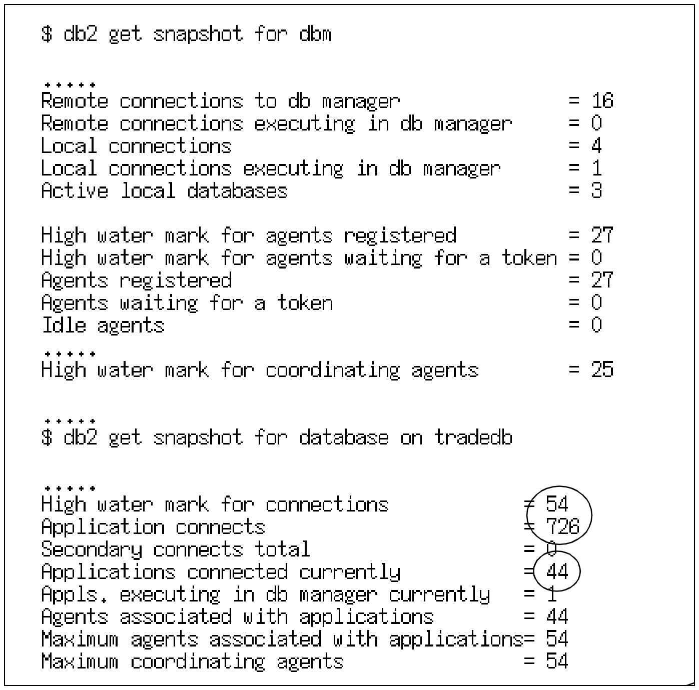
To determine whether connections were the cause of the problem, we listed both of the following items:

- Database manager configuration, and database configuration parameters, as shown in Figure 6-10 (edited to only show connection-related parameters)
- Database manager and database snapshots, as shown in Figure 6-11 on page 421 (edited to show monitoring elements of interest).



```
$ db2 get dbm cfg | grep -i agent
 Priority of agents                           (AGENTPRI) = 41
 Max number of existing agents               (MAXAGENTS) = 128
 Agent pool size                        (NUM_POOLAGENTS) = 64(calculated)
 Initial number of agents in pool       (NUM_INITAGENTS) = 0
 Max number of coordinating agents     (MAX_COORDAGENTS) = (MAXAGENTS - NUM_INITAGENTS)
 Max no. of concurrent coordinating agents  (MAXCAGENTS) = MAX_COORDAGENTS
 Max number of client connections      (MAX_CONNECTIONS) = MAX_COORDAGENTS
 SPM resync agent limit                 (SPM_MAX_RESYNC) = 20
$ db2 get db cfg for tradedb | grep -i appl
 Catalog cache size (4KB)              (CATALOGCACHE_SZ) = (MAXAPPLS*4)
 Max size of appl. group mem set (4KB) (APPGROUP_MEM_SZ) = 30000
 Percent of mem for appl. group heap   (GROUPHEAP_RATIO) = 70
 Max appl. control heap size (4KB)     (APP_CTL_HEAP_SZ) = 128
 Default application heap (4KB)             (APPLHEAPSZ) = 256
 Package cache size (4KB)                   (PCKCACHESZ) = (MAXAPPLS*8)
 Percent. of lock lists per application       (MAXLOCKS) = 10
 Max number of active applications            (MAXAPPLS) = 256
 Average number of active applications       (AVG_APPLS) = 1
 Max DB files open per application            (MAXFILOP) = 64
```

Figure 6-10 Database configuration connection-oriented parameters



```
$ db2 get snapshot for dbm

.....
Remote connections to db manager           = 16
Remote connections executing in db manager = 0
Local connections                          = 4
Local connections executing in db manager  = 1
Active local databases                     = 3

High water mark for agents registered      = 27
High water mark for agents waiting for a token = 0
Agents registered                          = 27
Agents waiting for a token                 = 0
Idle agents                                = 0
.....
High water mark for coordinating agents    = 25


.....
$ db2 get snapshot for database on tradedb


.....
High water mark for connections        = 54
Application connects                   = 726
Secondary connects total               = 0
Applications connected currently       = 44
Appls. executing in db manager currently = 1
Agents associated with applications    = 44
Maximum agents associated with applications= 54
Maximum coordinating agents            = 54
```

Figure 6-11 Snapshots to determine current connections

We concluded that connections were not the cause of the performance problem for the following reasons:

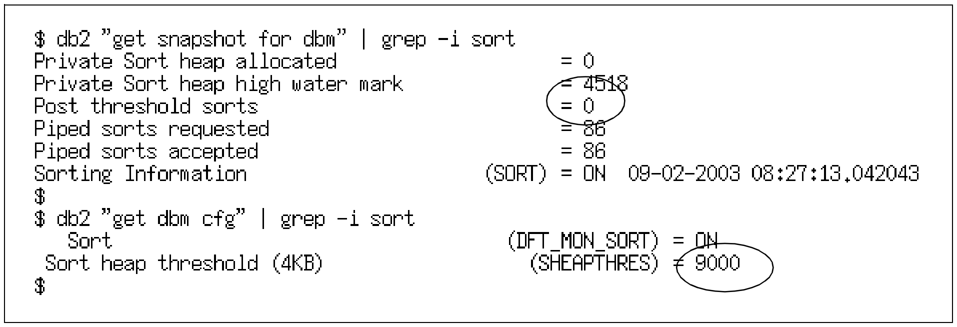
- Figure 6-10 on page 420 shows the following:
  - **Max number of existing agents** (`MAXAGENTS`) is 128.
  - **Max number of client connections** (`MAX_CONNECTIONS`) is 128.
  - **Max number of concurrent coordinating agents** (`MAXCAGENTS`) is 128.
  - Max number of active applications (`MAXAPPLS`) for the tradedb database is 256.
- Figure 6-11 shows the following:
  - The sum of (**Remote connections to db manager + Local connections**) is 20, which is less than `max_connections` (128).
  - The **High water mark for agents registered** (27) is less than `max_agents` (128).
  - The **High water mark for agents waiting for a token** is 0, which is less than `maxcagents` (128).
  - The **High water mark for connections** is 54, less than `maxappls` (256).

– The **High water mark for coordinating agents** is 25, less than **`max_coordagents`** (128).

Sorting problems

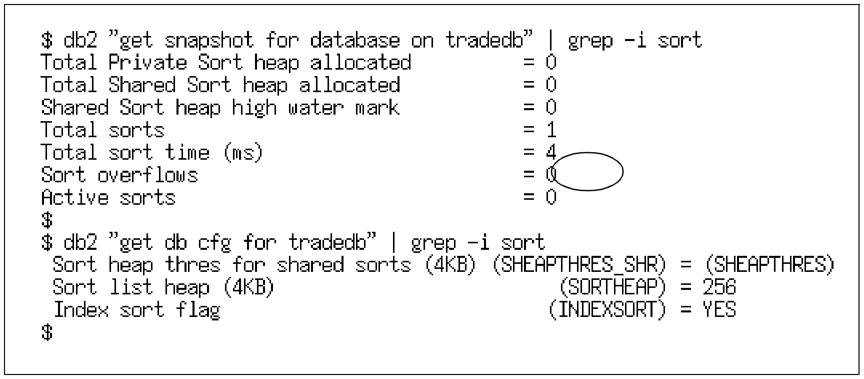
To determine whether sort considerations were the cause of the problem, we listed both of the following items:

- Database manager configuration parameters, and database manager snapshot, as shown in Figure 6-12 (edited to only show sort-related parameters
- Database configuration parameters, and database snapshot, as shown in Figure 6-13 (edited to show monitoring elements of interest).



```
$ db2 "get snapshot for dbm" | grep -i sort
Private Sort heap allocated                     = 0
Private Sort heap high water mark               = 4518
Post threshold sorts                            = 0
Piped sorts requested                           = 86
Piped sorts accepted                            = 86
Sorting Information                      (SORT) = ON  09-02-2003 08:27:13.042043
$
$ db2 "get dbm cfg" | grep -i sort
   Sort                                   (DFT_MON_SORT) = ON
 Sort heap threshold (4KB)                  (SHEAPTHRES) = 9000
$
```

Figure 6-12 Instance-level sort parameters and values



```
$ db2 "get snapshot for database on tradedb" | grep -i sort
Total Private Sort heap allocated          = 0
Total Shared Sort heap allocated           = 0
Shared Sort heap high water mark           = 0
Total sorts                                = 1
Total sort time (ms)                       = 4
Sort overflows                             = 0
Active sorts                               = 0
$
$ db2 "get db cfg for tradedb" | grep -i sort
 Sort heap thres for shared sorts (4KB) (SHEAPTHRES_SHR) = (SHEAPTHRES)
 Sort list heap (4KB)                         (SORTHEAP) = 256
 Index sort flag                             (INDEXSORT) = YES
$
```

Figure 6-13 Database-level sort parameters and values

We concluded that sorts were not the cause of the performance problem for the following reasons:

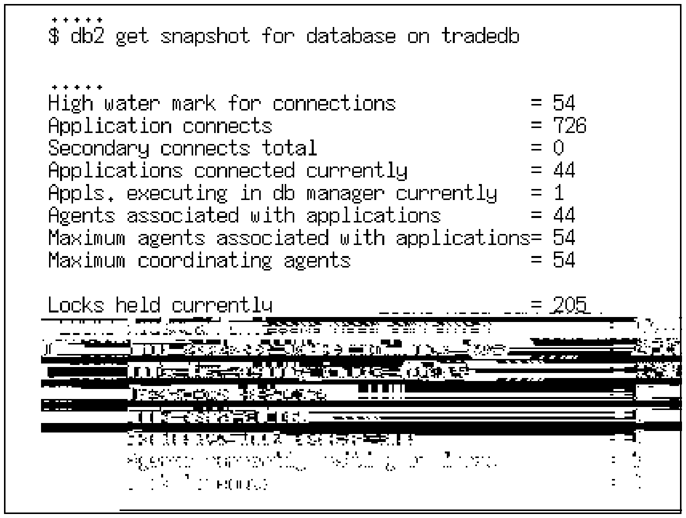
- Figure 6-12 on page 422 shows the following:
  - **Private Sort heap high water mark** (4518) is less than **`sheapthres`** (9000).
  - **Post threshold sorts** is 0.
  - The difference between **Piped sorts requested** and **Piped sorts accepted** is 0.
- Figure 6-13 on page 422 shows the following:
  - **Shared Sort heap high water mark** is 0.
  - **Sort overflows** is 0.

Note: Since this was a pure OLTP environment, we also verified that the **`INTRA_PARALLEL`** database manager configuration parameter was set to NO, using the following command:

```
   db2 get dbm cfg | grep -i intra
```

Locking problems

To determine whether locking considerations were the cause of the problem, we listed a database snapshot as shown in Figure 6-14 on page 424.



```
.....
$ db2 get snapshot for database on tradedb

.....
High water mark for connections           = 54
Application connects                      = 726
Secondary connects total                  = 0
Applications connected currently          = 44
Appls. executing in db manager currently  = 1
Agents associated with applications       = 44
Maximum agents associated with applications= 54
Maximum coordinating agents               = 54

Locks held currently                      = 205
[illegible]
```

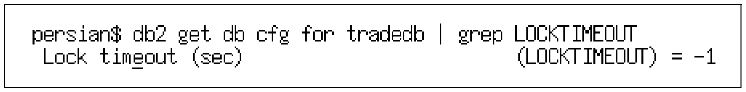
Figure 6-14 Database snapshot for locks

Figure 6-14 highlights the following areas of potential performance problems:

- There is a significant number of `Lock waits` (1223).
- `Agents currently waiting on locks` (8) is high for the number of `Applications connected currently` (44).
- The average time waiting on locks (`Time database waited on locks (ms)` / `Lock waits`) is (451510/1223) or 369 seconds, which is very high.

The lack of Lock Timeouts (`Lock Timeouts 0`), and lock escalations (`Lock escalations 0`) appears to be a good sign, except that the average time waiting for locks is too high. We would have expected to see at least a few lock timeouts for such a large wait time.

Since the recommendation for the database configuration parameter LOCKTIMEOUT for OLTP environments is usually less than 15 seconds, we decided to check the value of LOCKTIMEOUT, as shown in Figure 6-15.



```
persian$ db2 get db cfg for tradedb | grep LOCKTIMEOUT
 Lock timeout (sec)                        (LOCKTIMEOUT) = -1
```

Figure 6-15 Current value of LOCKTIMEOUT

A LOCKTIMEOUT value of -1 means that locks requests will never time out. Applications trying to acquire locks will wait indefinitely for the lock to be released by the holding application, which in our scenario does not occur. In the real world, applications with “infrequent commits” do relinquish locks, which enable the waiting applications to continue processing. However, the manifestation of such problems for applications is long waits and erratic response times.

Note: The default value of LOCKTIMEOUT is -1.

Root cause of the problem

Two factors contributed to the problem of extended waits:

1. Some applications did not commit frequently, thereby resulting in locks being held for extended periods of time.
2. The default LOCKTIMEOUT value of -1 caused applications requesting locks held by other applications to wait indefinitely.

Apply best practices

Two changes have to be made to resolve the problem as follows:

1. Change the LOCKTIMEOUT to a value between 10 and 15 seconds.

   Having too small a value can result in spurious timeouts, thereby frustrating users, while a very large value can cause extended waits that would also be unacceptable to users.

   Ongoing monitoring will help you to determine the appropriate value for your environment.
2. Identify applications that execute in multi-user shared data environments that hold large locks (table level) and commit infrequently, and encourage the application developer to reduce the size and duration of locks held.

   However, in some cases, business requirements may not permit this, in which case, you should consider scheduling these applications during time periods that result in the least amount of contention.

Note: We did not attempt to identify the application that was committing infrequently in this scenario.

By not doing so, and just resolving the LOCKTIMEOUT issue by setting it to an appropriate value, we merely deferred the “infrequent commits” problem (namely, the certainty of another triggering event that users will be experiencing a large number of timeouts in future).

In real world environments, however, it is critical that both factors be addressed to resolve the problem scenario modeled here.

6.3.2 Poor SQL performance due to missing indexes (OLTP)

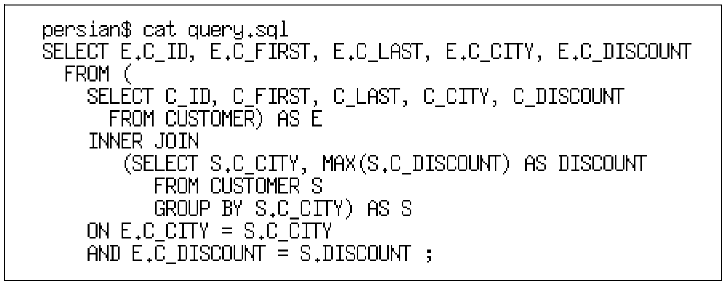
The scenario described here is one of diagnosing a performance problem caused by missing indexes.

Missing indexes may affect the choice of an optimal access path, and may result in table scans and/or sorts resulting in poor SQL performance. The reasons for a missing index may be due to an inadvertent dropping of the index, or an ignorance of the application workload's characteristics during the design phase, which resulted in the indexes not being defined.

Description of the application

We used the DTW application described in Appendix B, "Workloads used in the scenarios" on page 485, which is a slightly modified version of the industry standard TPC-C OLTP benchmark.

We also created a custom query against the DTW tables that would result in a suboptimal access path because of the absence of an index. This query is shown in Figure 6-16, and it identifies the customer with the maximum discount in each city.



```
persian$ cat query.sql
SELECT E.C_ID, E.C_FIRST, E.C_LAST, E.C_CITY, E.C_DISCOUNT
  FROM (
    SELECT C_ID, C_FIRST, C_LAST, C_CITY, C_DISCOUNT
      FROM CUSTOMER) AS E
    INNER JOIN
       (SELECT S.C_CITY, MAX(S.C_DISCOUNT) AS DISCOUNT
          FROM CUSTOMER S
          GROUP BY S.C_CITY) AS S
    ON E.C_CITY = S.C_CITY
    AND E.C_DISCOUNT = S.DISCOUNT ;
```

Figure 6-16 Custom query for missing index scenario

Environment configuration

We used a single pSeries Model 270 running the DB2 database; this workload did not have WebSphere Application Server support.

Table 6-3 on page 427 and Figure 6-17 on page 427 highlight the details of the environment.

Table 6-3 Missing indexes scenario configuration

Hardware configuration	Software configuration
pSeries Model 270 2 CPUs 1 GB RAM 2 x 36 GB SCSI disks	AIX 5.1 ML03 DB2 V8.1 FP1
PC	Windows 2000

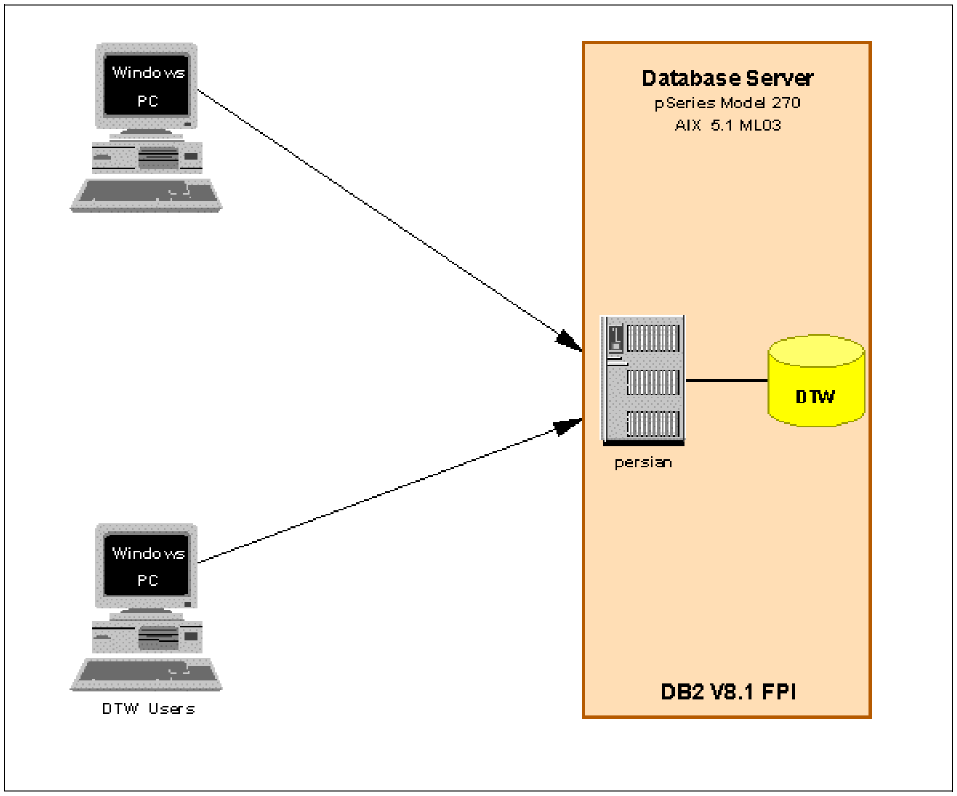


Figure 6-17 Missing indexes configuration environment

Monitor level settings

We used the following recommended settings for routine monitoring in an OLTP environment, as shown in Figure 6-6 on page 415:

- Set monitor switches **`DFT_MON_BUFPOOL`**, **`DFT_MON_SORT`**, **`DFT_MON_TABLE`**, **`DFT_MON_TIMESTAMP`**, **`DFT_MON_UOW`**, and **`HEALTH_MON`** to ON.
- Set monitor switches **`DFT_MON_LOCK`** and **`DFT_MON_STMT`** to OFF.
- Accept the **`DIAGLEVEL`** default setting (3).

Workload used

We used several DTW driver programs to perform a variety of queries, updates, inserts, and deletes against the DTW database. This provided the background OLTP workload, and we ran our custom query shown in Figure 6-16 on page 426 to create the performance problem.

Triggering event

Several user complaints about an application that was experiencing long response times for a recently introduced application.

Hypotheses and validations

We postulated the following hypotheses as potential causes of the problem and then attempted to validate each hypothesis in turn:

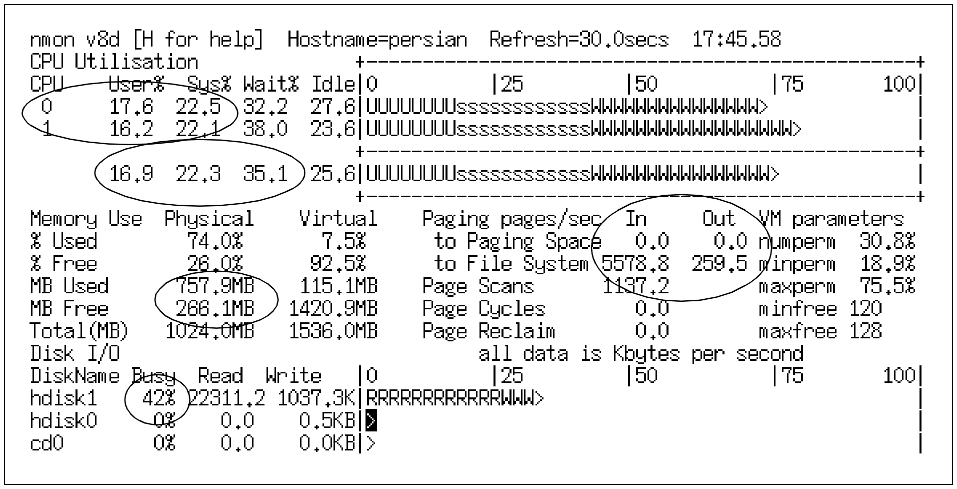
- Hypothesis 1: Network performance problems
- Hypothesis 2: DB2 database server system problems
- Hypothesis 3: DB2 system problems
- Hypothesis 4: DB2 application problems

Hypothesis 1: Network performance problems

Using the same approach described in "Hypothesis 1: Network performance problems" on page 416, we determined that network availability and bandwidth was most likely not the cause of the performance problem.

Hypothesis 2: DB2 database server system problems

Figure 6-18 on page 429 shows the output of the **`nmon`** command while the complaints were being received about the poorly performing application.



```
nmon v8d [H for help]  Hostname=persian  Refresh=30.0secs  17:45.58
CPU Utilisation               +-------------------------------------------------+
CPU    User%  Sys% Wait% Idle|0          |25         |50          |75       100|
 0      17.6  22.5  32.2  27.6|UUUUUUUUsssssssssssWWWWWWWWWWWWWWW>               |
 1      16.2  22.1  38.0  23.6|UUUUUUUUsssssssssssWWWWWWWWWWWWWWWWWW>            |
                              +-------------------------------------------------+
        16.9  22.3  35.1  25.6|UUUUUUUUsssssssssssWWWWWWWWWWWWWWW>               |
                              +-------------------------------------------------+
Memory Use  Physical   Virtual    Paging pages/sec  In     Out  VM parameters
% Used         74.0%      7.5%     to Paging Space   0.0    0.0 numperm  30.8%
% Free         26.0%     92.5%     to File System 5578.8  259.5 minperm  18.9%
MB Used      757.9MB   115.1MB    Page Scans      1137.2        maxperm  75.5%
MB Free      266.1MB  1420.9MB    Page Cycles        0.0        minfree 120
Total(MB)   1024.0MB  1536.0MB    Page Reclaim       0.0        maxfree 128
Disk I/O                               all data is Kbytes per second
DiskName Busy  Read  Write   |0          |25         |50          |75       100|
hdisk1    42% 22311.2 1037.3K|RRRRRRRRRRRRWWW>                                  |
hdisk0     0%     0.0    0.5KB|>                                                |
cd0        0%     0.0    0.0KB|>                                                |
```

Figure 6-18 System load while running application

We concluded that the DB2 system was not the cause of the performance problem for the following reasons:

- ► While the system utilization is towards the upper end of the acceptable range at 75%, there are still spare CPU cycles available.
- ► The workload is relatively evenly balanced across the two CPUs; therefore, the problem is unlikely to be a runaway or looping application or other process.
- ► The system has 266 MB of free memory and is not paging; therefore, memory is not a problem.
- ► One disk (hdisk1) is relatively busier than the others, and it is worth investigating whether the I/O load should be balanced across multiple disks. However, the disk utilization of 42% is well within the recommended bounds, so we can therefore conclude that the disk subsystem is not likely to be a performance bottleneck.

Hypothesis 3: DB2 system problems

Using the same hypotheses hierarchy described in "DB2 system resource constraints" on page 400, we validated the following in sequence:

- ► Connection problems
- ► Sorting problems
- ► Locking problems
- ► Buffer pool problems
- ► Cache size problems
- ► Miscellaneous problems

Connection problems

Using the same approach described in “Hypothesis 4: DB2 system problems” on page 419, we determined that connection problems were not the cause of the performance problem.

Sorting problems

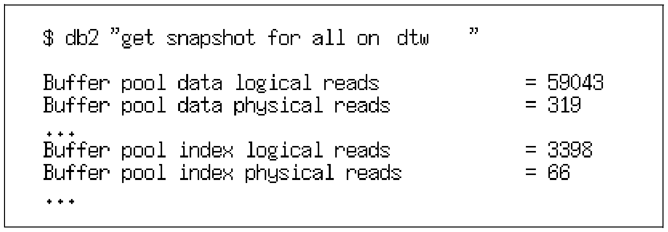
Using the same approach described in “Hypothesis 4: DB2 system problems” on page 419, we determined that sorting problems were not the cause of the performance problem

Locking problems

Using the same approach described in “Hypothesis 4: DB2 system problems” on page 419, we determined that locking problems were not the cause of the performance problem.

Buffer pool problems

Figure 6-19 shows the snapshot data relevant to buffer pools for the DTW database.



```
$ db2 "get snapshot for all on dtw   "

Buffer pool data logical reads            = 59043
Buffer pool data physical reads           = 319
...
Buffer pool index logical reads           = 3398
Buffer pool index physical reads          = 66
...
```

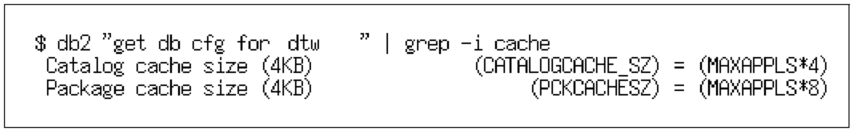
Figure 6-19 Buffer pool snapshot data

We concluded that the buffer pool was not the cause of the performance problem since the hit ratios were well within acceptable limits, as follows:

- Overall buffer pool hit ratio is (1-((319+66)/(59043+3398)))*100%, which is approximately 100%.
- Index buffer pool hit ratio is (1-(66/3398))*100%, which is about 98%.

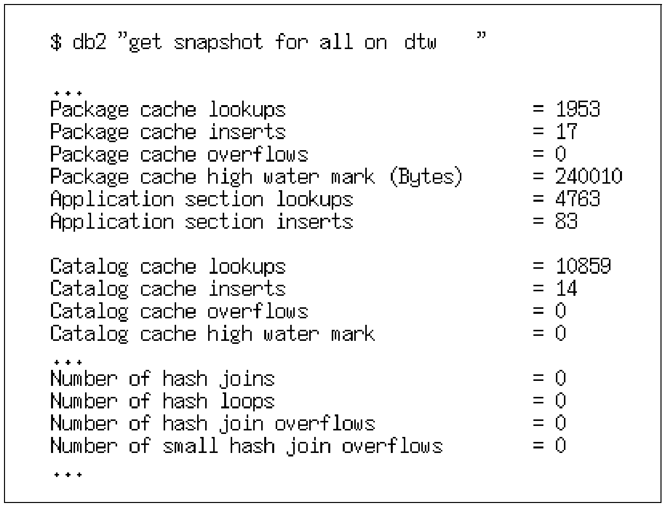
Cache size problems

To determine whether catalog and package cache size considerations were the cause of the problem, we listed the database configuration parameters and the database snapshot, as shown in Figure 6-20 on page 431 and Figure 6-21 on page 431, respectively (edited to show monitoring elements of interest).



```
$ db2 "get db cfg for  dtw    " | grep -i cache
 Catalog cache size (4KB)              (CATALOGCACHE_SZ) = (MAXAPPLS*4)
 Package cache size (4KB)                   (PCKCACHESZ) = (MAXAPPLS*8)
```

Figure 6-20   Database package and catalog cache sizes



```
$ db2 "get snapshot for all on  dtw    "

...
Package cache lookups                      = 1953
Package cache inserts                      = 17
Package cache overflows                    = 0
Package cache high water mark (Bytes)      = 240010
Application section lookups                = 4763
Application section inserts                = 83

Catalog cache lookups                      = 10859
Catalog cache inserts                      = 14
Catalog cache overflows                    = 0
Catalog cache high water mark              = 0
...
Number of hash joins                       = 0
Number of hash loops                       = 0
Number of hash join overflows              = 0
Number of small hash join overflows        = 0
...
```

Figure 6-21   Package and catalog cache snapshot data

We concluded that there were no cache size problems, for the following reasons:

- Package cache hit ratio is (1-(17/1953))* 100%, which is 99%.
- Package cache overflows is 0.
- Package cache high water mark is 240010, which is less than the Package cache size (PCKCACHESZ) (256*4*4 KB or 4 MB).
- Catalog cache hit ratio is (1-(14/10859)) * 100%, which is 99.8%.
- Catalog cache overflows is 0.
- Catalog cache high water mark (0) is less than Catalog cache size (CATALOGCACHE_SZ).

Miscellaneous problems

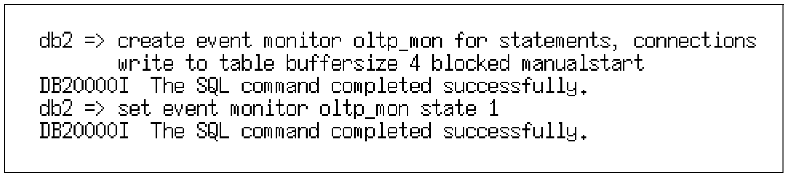
Given our controlled environments, we chose not to monitor the various miscellaneous constraints listed in “Miscellaneous constraints” on page 410. In the real world however, these should be evaluated and eliminated as contributing to poor performance.

Hypothesis 4: DB2 application problems

Having eliminated the network, the DB2 system, and the DB2 server as being the potential cause of the performance problem, we turned our attention to the application.

We needed to identify the errant application and SQL, and then analyze the SQL in question.

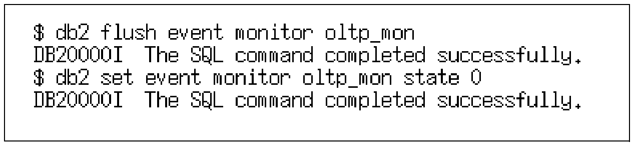
To identify the application, we created an Event Monitor for connections and statements, and activated it at the beginning of the monitoring interval by changing its state to 1 as shown in Figure 6-22.



```
db2 => create event monitor oltp_mon for statements, connections
       write to table buffersize 4 blocked manualstart
DB20000I  The SQL command completed successfully.
db2 => set event monitor oltp_mon state 1
DB20000I  The SQL command completed successfully.
```

Figure 6-22   Creating the Event Monitor

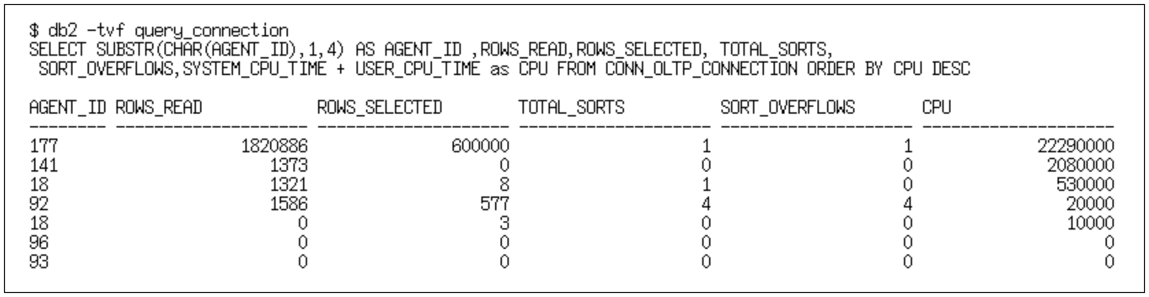
We then had to wait for the next monitoring interval when users reran their applications, in order to collect Event Monitor data. Once the applications completed, we flushed and stopped the Event Monitors, as shown in Figure 6-23.



```
$ db2 flush event monitor oltp_mon
DB20000I  The SQL command completed successfully.
$ db2 set event monitor oltp_mon state 0
DB20000I  The SQL command completed successfully.
```

Figure 6-23   Flushing Event Monitor data to tables

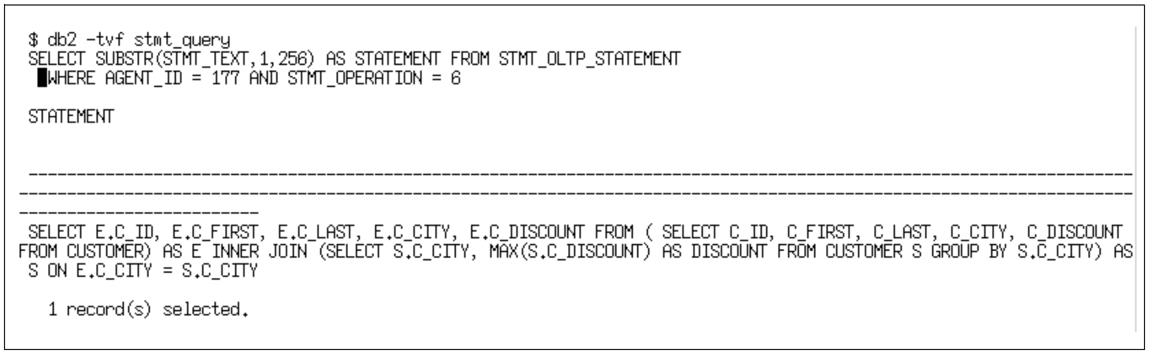
The Event Monitor data is written to SQL tables, and we queried the contents of the connection Event Monitor table to find the query using the most CPU cycles (assuming this was the poorly performing application) as shown in Figure 6-24 on page 433. This also revealed that AGENT_ID 177 was reading a large number (1820886) of rows.



```
$ db2 -tvf query_connection
SELECT SUBSTR(CHAR(AGENT_ID),1,4) AS AGENT_ID ,ROWS_READ,ROWS_SELECTED, TOTAL_SORTS,
 SORT_OVERFLOWS,SYSTEM_CPU_TIME + USER_CPU_TIME as CPU FROM CONN_OLTP_CONNECTION ORDER BY CPU DESC

AGENT_ID ROWS_READ            ROWS_SELECTED        TOTAL_SORTS          SORT_OVERFLOWS       CPU
-------- -------------------- -------------------- -------------------- -------------------- --------------------
177                   1820886               600000                    1                    1             22290000
141                      1373                    0                    0                    0              2080000
18                       1321                    8                    1                    0               530000
92                       1586                  577                    4                    4                20000
18                          0                    3                    0                    0                10000
96                          0                    0                    0                    0                    0
93                          0                    0                    0                    0                    0
```

Figure 6-24 Write to table Event Monitor data showing CPU usage and so on

Using the agent_id (177) of the long running query, we then queried the statement Event Monitor table to obtain the SQL statement details, as shown in Figure 6-25.

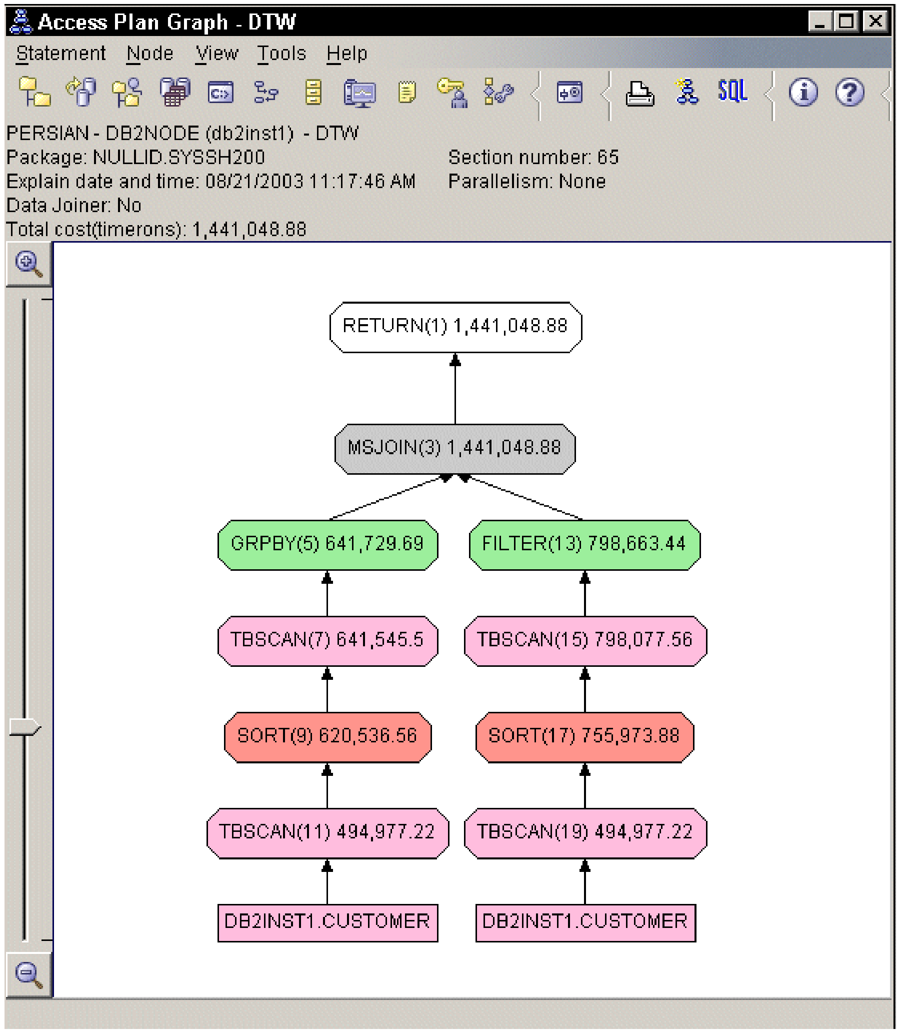


```
$ db2 -tvf stmt_query
SELECT SUBSTR(STMT_TEXT,1,256) AS STATEMENT FROM STMT_OLTP_STATEMENT
 WHERE AGENT_ID = 177 AND STMT_OPERATION = 6

STATEMENT


--------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------
------------------------
 SELECT E.C_ID, E.C_FIRST, E.C_LAST, E.C_CITY, E.C_DISCOUNT FROM ( SELECT C_ID, C_FIRST, C_LAST, C_CITY, C_DISCOUNT
FROM CUSTOMER) AS E INNER JOIN (SELECT S.C_CITY, MAX(S.C_DISCOUNT) AS DISCOUNT FROM CUSTOMER S GROUP BY S.C_CITY) AS
 S ON E.C_CITY = S.C_CITY

  1 record(s) selected.
```

Figure 6-25 Write to table Event Monitor data showing SQL statement

We concluded that the SQL statement thus identified was problematic since a large number of rows were being read, and CPU consumption was high.

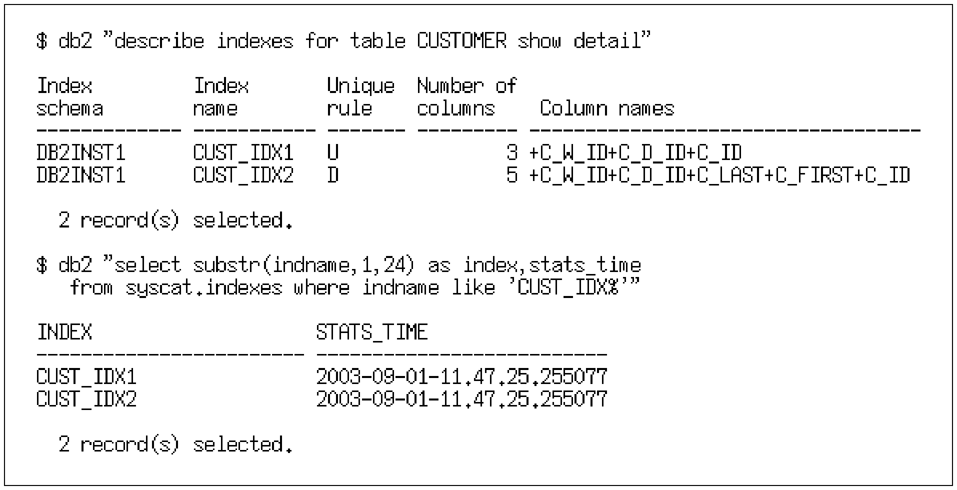
We then checked to see if there were access path problems with this SQL statement by running Visual Explain on it as shown in Figure 6-26 on page 434, which showed that the query was performing table scans.



PERSIAN - DB2NODE (db2inst1) - DTW
Package: NULLID.SYSSH200
Section number: 65
Explain date and time: 08/21/2003 11:17:46 AM
Parallelism: None
Data Joiner: No
Total cost(timerons): 1,441,048.88

Figure 6-26 Visual Explain output

We then checked to see what indexes existed on the CUSTOMER table as shown in Figure 6-27 on page 435, and when statistics were last collected on them.

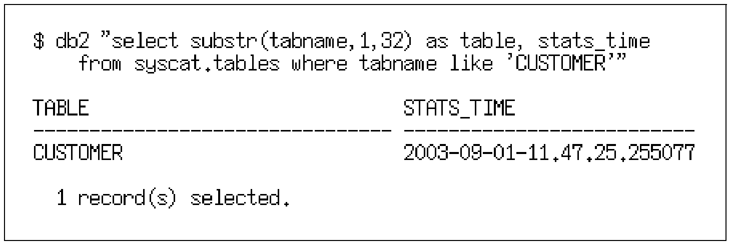


```
$ db2 "describe indexes for table CUSTOMER show detail"

Index         Index       Unique  Number of
schema        name        rule    columns    Column names
------------- ----------- ------- --------- -----------------------------------
DB2INST1      CUST_IDX1   U               3 +C_W_ID+C_D_ID+C_ID
DB2INST1      CUST_IDX2   D               5 +C_W_ID+C_D_ID+C_LAST+C_FIRST+C_ID

  2 record(s) selected.

$ db2 "select substr(indname,1,24) as index,stats_time
   from syscat.indexes where indname like 'CUST_IDX%'"

INDEX                    STATS_TIME
------------------------ --------------------------
CUST_IDX1                2003-09-01-11.47.25.255077
CUST_IDX2                2003-09-01-11.47.25.255077

  2 record(s) selected.
```

Figure 6-27 Indexes on the CUSTOMER table

While indexes exist, there are no indexes for the column `C_CITY`, which is the join predicate of the SQL statement.

We also checked to see when statistics were last collected on the CUSTOMER table, as shown in Figure 6-28.



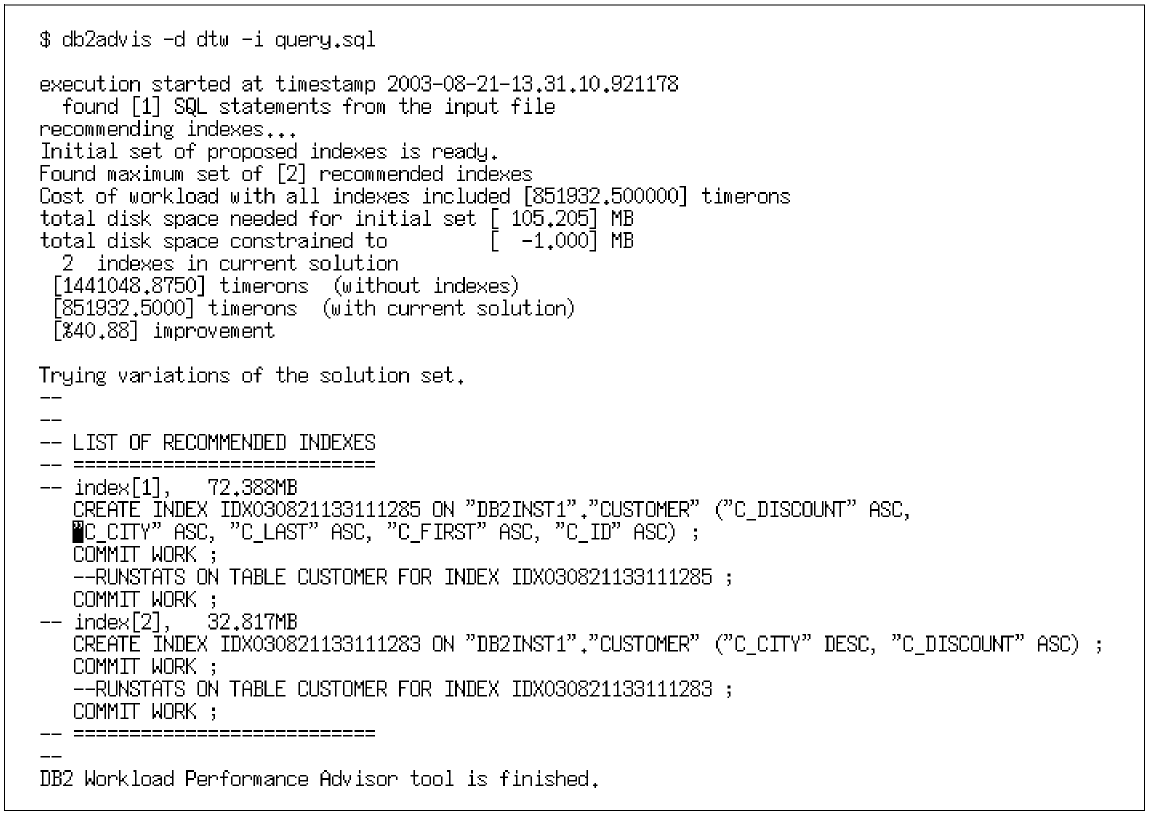
```
$ db2 "select substr(tabname,1,32) as table, stats_time
    from syscat.tables where tabname like 'CUSTOMER'"

TABLE                            STATS_TIME
-------------------------------- --------------------------
CUSTOMER                         2003-09-01-11.47.25.255077

  1 record(s) selected.
```

Figure 6-28 Checking when runstats last ran on the customer table

Since statistics were current and indexes were not being used, we invoked the Design Advisor for this query, as shown in Figure 6-29 on page 436, to determine whether new indexes would improve the query performance.

Attention: If the statistics were found to be missing or out of date, then **`runstats`** would have to be run, and the query EXPLAINed and re-executed to determine the cause of the performance problem.

Another validation (not shown here) is the necessity to verify that the tables involved in a query are well organized. Use the **`reorgchk`** utility to determine this, and run **`reorg`** and **`runstats`** if the table is disorganized, before rerunning the query.



```
$ db2advis -d dtw -i query.sql

execution started at timestamp 2003-08-21-13.31.10.921178
  found [1] SQL statements from the input file
recommending indexes...
Initial set of proposed indexes is ready.
Found maximum set of [2] recommended indexes
Cost of workload with all indexes included [851932.500000] timerons
total disk space needed for initial set [ 105.205] MB
total disk space constrained to         [  -1.000] MB
  2  indexes in current solution
 [1441048.8750] timerons  (without indexes)
 [851932.5000] timerons  (with current solution)
 [%40.88] improvement

Trying variations of the solution set.
--
--
-- LIST OF RECOMMENDED INDEXES
-- ===========================
-- index[1],   72.388MB
   CREATE INDEX IDX030821133111285 ON ”DB2INST1”.”CUSTOMER” (”C_DISCOUNT” ASC,
   ”C_CITY” ASC, ”C_LAST” ASC, ”C_FIRST” ASC, ”C_ID” ASC) ;
   COMMIT WORK ;
   --RUNSTATS ON TABLE CUSTOMER FOR INDEX IDX030821133111285 ;
   COMMIT WORK ;
-- index[2],   32.817MB
   CREATE INDEX IDX030821133111283 ON ”DB2INST1”.”CUSTOMER” (”C_CITY” DESC, ”C_DISCOUNT” ASC) ;
   COMMIT WORK ;
   --RUNSTATS ON TABLE CUSTOMER FOR INDEX IDX030821133111283 ;
   COMMIT WORK ;
-- ===========================
--
DB2 Workload Performance Advisor tool is finished.
```

Figure 6-29 Design Advisor recommendations

Root cause of the problem

The root cause of the poor performance of this application is missing indexes that resulted in suboptimal access path.

Apply best practices

Creating the indexes recommended by Design Advisor appears to be beneficial, based on projections of a 40.88% improvement. You need to create the indexes and measure actual performance gains.

Creating new indexes consumes disk space, and will negatively impact applications performing inserts, updates, and deletes due to index maintenance. Additionally, index contention may also affect performance.

6.3.3 Poor SQL performance due to unused MQTs (BI)

The scenario described here is one of diagnosing a problem of a query not using a Materialized Query Table (MQT).

MQTs provide a powerful mechanism for achieving superior query performance in a BI environment. They typically contain precomputed aggregates from one or more tables and are generally maintained automatically by DB2. Whenever possible, the DB2 optimizer will automatically rewrite a query accessing large base tables to use an MQT instead in order to deliver superior performance.

Description of the application

We used the EBIZ banking application described in Appendix B.3, “EBIZ database” on page 486 as an example of databases used in a BI environment.

Environment configuration

We used a single pSeries Model 650 8 CPU system to running the DB2 database; this workload did not have WebSphere Application Server support.

Table 6-4 and Figure 6-30 on page 438 highlight the details of the environment.

Table 6-4 Unused MQTs scenario configuration

Hardware configuration	Software configuration
pSeries Model 650 8 CPUs 32 GB RAM 4 x 146 GB HDD disks	AIX 5.1 ML03 DB2 V8.1 FP1
PC	Windows 2000

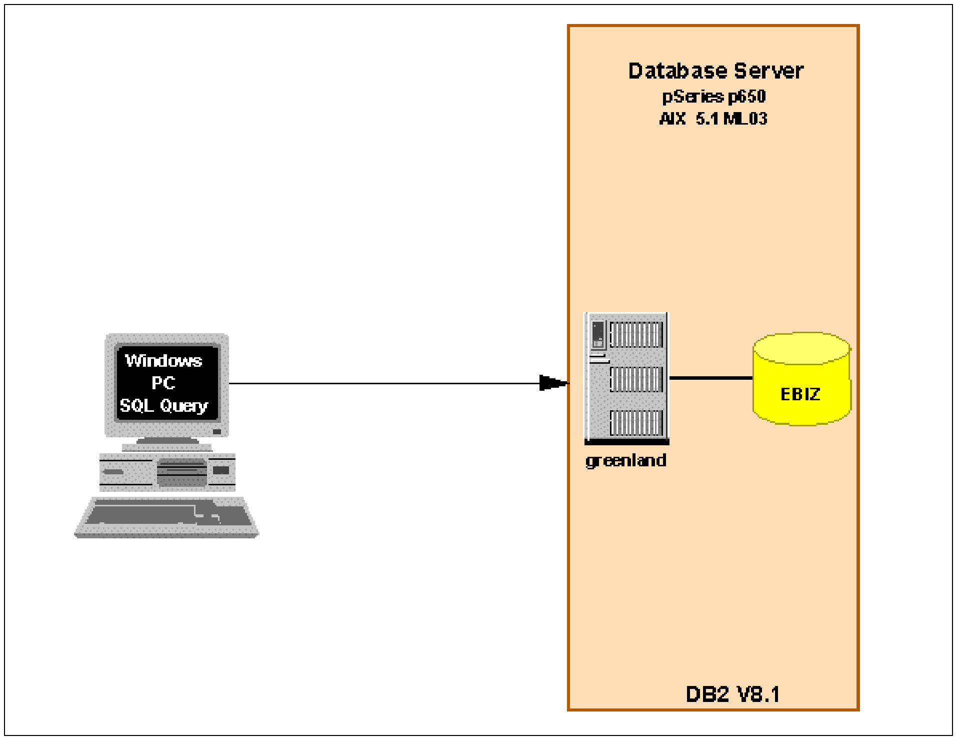


Figure 6-30 Unused MQTs configuration environment

Monitor level settings

We used the following recommended settings for routine monitoring in a BI environment as shown in Figure 6-30:

- Set monitor switches **DFT_MON_BUFPOOL**, **DFT_MON_SORT**, **DFT_MON_TABLE**, **DFT_MON_TIMESTAMP**, and **HEALTH_MON** to ON.
- Set monitor switches **DFT_MON_LOCK**, **DFT_MON_STMT** and **DFT_MON_UOW** to OFF.
- Accept the **DIAGLEVEL** default setting (3).

Workload used

The workload consisted of several BI SQL queries against the EBIZ database.

Triggering event

Several user complaints about very poor response times for a newly introduced set of queries.

Hypotheses and validations

We postulated the following hypotheses as potential causes of the problem, and then attempted to validate each hypothesis in turn:

- Hypothesis 1: Network performance problems
- Hypothesis 2: DB2 database server system problems
- Hypothesis 3: DB2 system problems
- Hypothesis 4: DB2 application problems

Hypothesis 1: Network performance problems

Using the same approach described in “Hypothesis 1: Network performance problems” on page 416, we determined that network availability and bandwidth was most likely not the cause of the performance problem.

Hypothesis 2: DB2 database server system problems

Using the same approach described in “Hypothesis 2: DB2 database server system problems” on page 428, we determined that system CPU, I/O, and memory was not the cause of the performance problem.

Hypothesis 3: DB2 system problems

Using the same approach described in “Hypothesis 3: DB2 system problems” on page 429, we concluded that connections, sorting, locking, buffer pool, and cache sizes were not the cause of the performance problems.

Hypothesis 4: Application problem

Having eliminated the network, DB2 system and DB2 server as being the potential cause of the performance problem, we turned our attention to the application.

We needed to identify the errant application and SQL, and then analyze the SQL in question. To identify the application, we created an Event Monitor for connections and statements, and activated it at the beginning of the monitoring interval by changing its state to 1, as shown in Figure 6-31 on page 440. We also identified in which tables the Event Monitors would store their results.

```
$ db2 -tvf create_evmon.sql
create event monitor mqt_connection for connections write to table
 buffersize 4 blocked manualstart
DB20000I  The SQL command completed successfully.

create event monitor mqt_statement for statements write to table
 buffersize 4 blocked manualstart
DB20000I  The SQL command completed successfully.

set event monitor mqt_connection state 1
DB20000I  The SQL command completed successfully.

set event monitor mqt_statement state 1
DB20000I  The SQL command completed successfully.

$ db2 -tvf query_evmon.sql
select substr(evmonname,1,18) as event_monitor, substr(tabname,1,32) as table_name
 from syscat.eventtables where evmonname like 'MQT_S%' or evmonname like 'MQT_C%'

EVENT_MONITOR      TABLE_NAME
------------------ --------------------------------
MQT_CONNECTION     CONN_MQT_CONNECTION
MQT_CONNECTION     CONNHEADER_MQT_CONNECTION
MQT_CONNECTION     CONTROL_MQT_CONNECTION
MQT_STATEMENT      STMT_MQT_STATEMENT_6
MQT_STATEMENT      CONNHEADER_MQT_STATEMENT
MQT_STATEMENT      CONTROL_MQT_STATEMENT

  6 record(s) selected.
```

Figure 6-31 Event monitors for unused MQT scenario

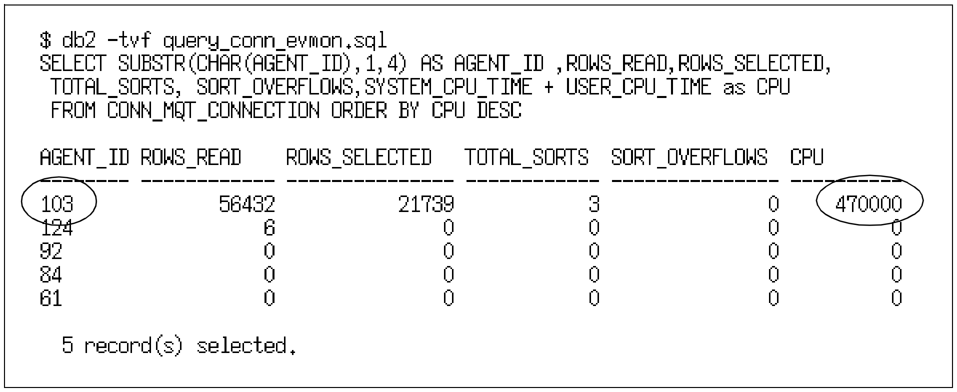
We then had to wait for the next monitoring interval when users reran their applications in order to collect Event Monitor data.

We executed a **`db2 list application`** command to view the running applications, as shown in Figure 6-32.

```
$ db2 list applicationn

Auth Id  Application    Appl.      Application Id                 DB       # of
         Name           Handle                                    Name     Agents
-------- -------------- ---------- ------------------------------ -------- -----
DB2INST1 db2bp          124        *LOCAL.db2inst1.08C332185950   EBIZ     1
DB2INST1 db2bp          103        *LOCAL.db2inst1.05B472175613   EBIZ     1
DB2INST1 db2bp          92         *LOCAL.db2inst1.0609D2171229   EBIZ     1
DB2INST1 javaw.exe      84         G901278B.G90C.00D101220751     EBIZ     1
DB2INST1 javaw.exe      61         G901278B.E70B.00D101200905     EBIZ     1
```

Figure 6-32 Applications running during unused MQT scenario

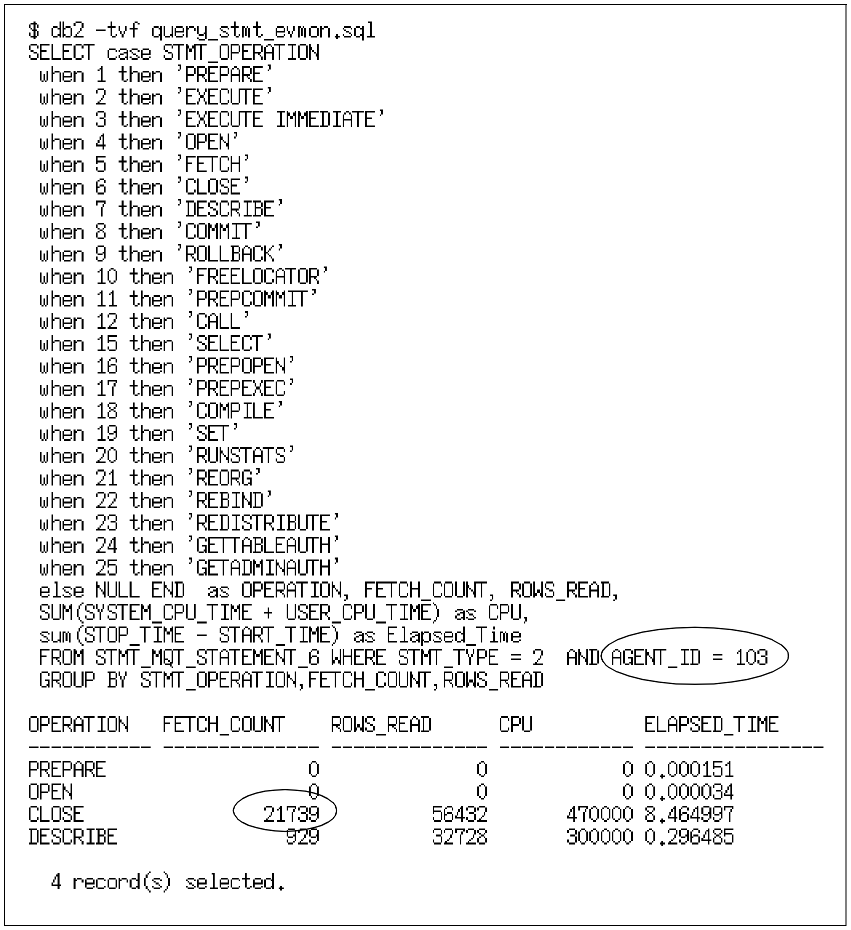
After the applications finished, we flushed and stopped the Event Monitors and examined the Event Monitor connection data, as shown in Figure 6-33 on page 441.



```
$ db2 -tvf query_conn_evmon.sql
SELECT SUBSTR(CHAR(AGENT_ID),1,4) AS AGENT_ID ,ROWS_READ,ROWS_SELECTED,
 TOTAL_SORTS, SORT_OVERFLOWS,SYSTEM_CPU_TIME + USER_CPU_TIME as CPU
 FROM CONN_MQT_CONNECTION ORDER BY CPU DESC

AGENT_ID ROWS_READ    ROWS_SELECTED   TOTAL_SORTS  SORT_OVERFLOWS  CPU
-------- ------------ --------------- ------------ --------------- ----------
103             56432           21739            3               0     470000
124                 6               0            0               0          0
92                  0               0            0               0          0
84                  0               0            0               0          0
61                  0               0            0               0          0

  5 record(s) selected.
```

Figure 6-33 Connection Event Monitor output for unused MQT scenario

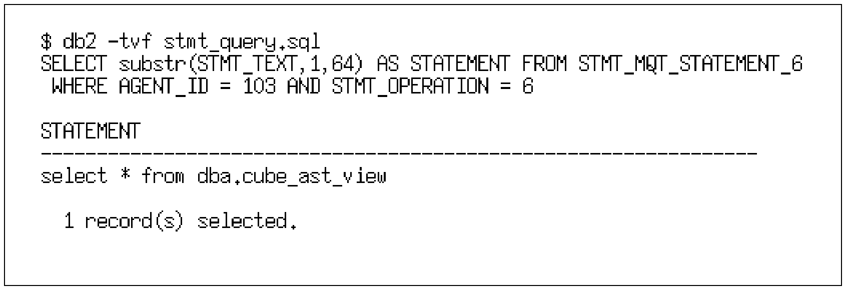
We noted that AGENT_ID 103 consumed a large amount of CPU and read a large number of rows. We confirmed that this was the problem query by selecting the data for agent_id 103 from the statement Event Monitor table, as shown in Figure 6-34 on page 442.



```
$ db2 -tvf query_stmt_evmon.sql
SELECT case STMT_OPERATION
 when 1 then 'PREPARE'
 when 2 then 'EXECUTE'
 when 3 then 'EXECUTE IMMEDIATE'
 when 4 then 'OPEN'
 when 5 then 'FETCH'
 when 6 then 'CLOSE'
 when 7 then 'DESCRIBE'
 when 8 then 'COMMIT'
 when 9 then 'ROLLBACK'
 when 10 then 'FREELOCATOR'
 when 11 then 'PREPCOMMIT'
 when 12 then 'CALL'
 when 15 then 'SELECT'
 when 16 then 'PREPOPEN'
 when 17 then 'PREPEXEC'
 when 18 then 'COMPILE'
 when 19 then 'SET'
 when 20 then 'RUNSTATS'
 when 21 then 'REORG'
 when 22 then 'REBIND'
 when 23 then 'REDISTRIBUTE'
 when 24 then 'GETTABLEAUTH'
 when 25 then 'GETADMINAUTH'
 else NULL END  as OPERATION, FETCH_COUNT, ROWS_READ,
 SUM(SYSTEM_CPU_TIME + USER_CPU_TIME) as CPU,
 sum(STOP_TIME - START_TIME) as Elapsed_Time
 FROM STMT_MQT_STATEMENT_6 WHERE STMT_TYPE = 2  AND AGENT_ID = 103
 GROUP BY STMT_OPERATION,FETCH_COUNT,ROWS_READ

OPERATION   FETCH_COUNT    ROWS_READ      CPU          ELAPSED_TIME
----------- -------------- -------------- ------------ ----------------
PREPARE                  0              0            0 0.000151
OPEN                     0              0            0 0.000034
CLOSE                21739          56432       470000 8.464997
DESCRIBE               929          32728       300000 0.296485

  4 record(s) selected.
```

Figure 6-34   Statement Event Monitor output for unused MQT scenario

We extracted the corresponding SQL statement from the statement Event Monitor output, as shown in Figure 6-35 on page 443.



```
$ db2 -tvf stmt_query.sql
SELECT substr(STMT_TEXT,1,64) AS STATEMENT FROM STMT_MQT_STATEMENT_6
 WHERE AGENT_ID = 103 AND STMT_OPERATION = 6

STATEMENT
----------------------------------------------------------------
select * from dba.cube_ast_view

  1 record(s) selected.
```

Figure 6-35   Statement Event Monitor data for unused MQT scenario - SQL

We then checked to see if there were access path problems with this SQL statement by running Visual Explain on it as shown in Figure 6-36 on page 444, which showed that the query was performing multiple table scans.

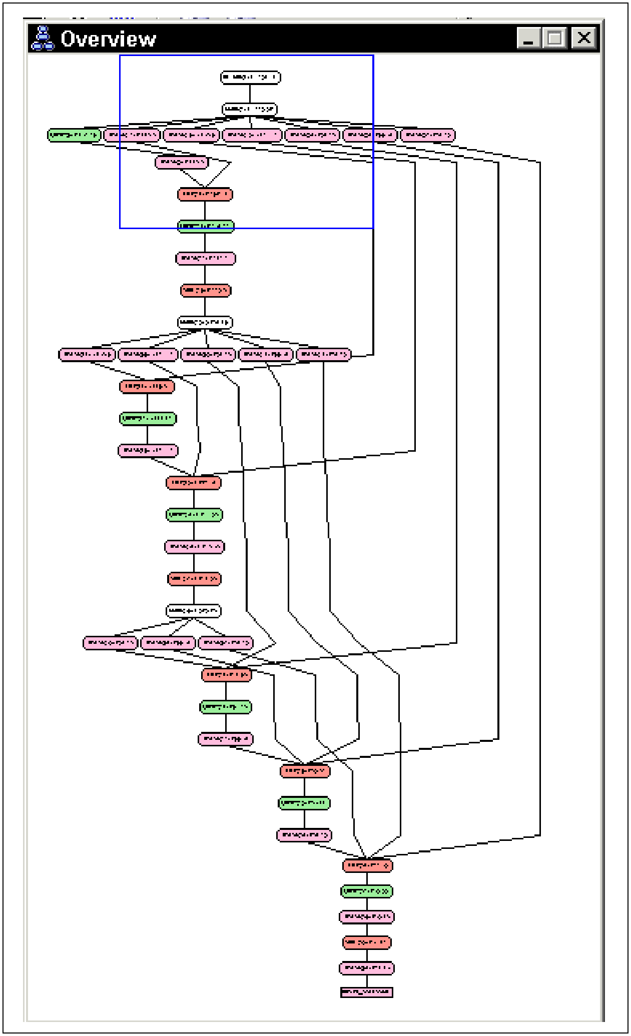
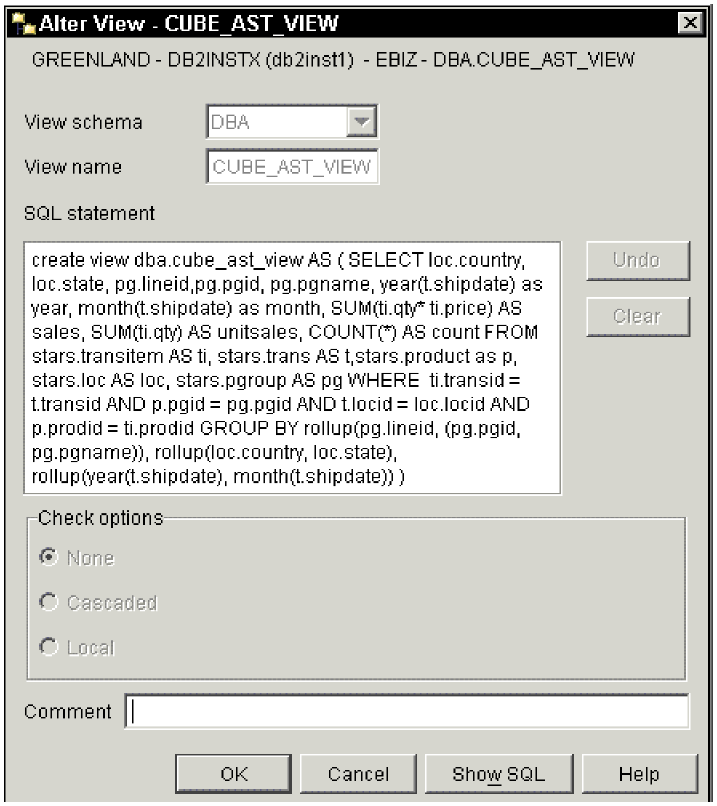


Figure 6-36 Visual Explain output for unused MQT scenario

We therefore looked at the table dba.cust_ast_view in more detail, and concluded that was a view. Using Control Center, we obtained the view definition as shown in Figure 6-37 on page 445.


Alter View - CUBE_AST_VIEW
GREENLAND - DB2INSTX (db2inst1) - EBIZ - DBA.CUBE_AST_VIEW
Undo
Clear
OK
Cancel
Show SQL
Help

View schema: DBA

View name: CUBE_AST_VIEW

SQL statement

```
create view dba.cube_ast_view AS ( SELECT loc.country,
loc.state, pg.lineid,pg.pgid, pg.pgname, year(t.shipdate) as
year, month(t.shipdate) as month, SUM(ti.qty* ti.price) AS
sales, SUM(ti.qty) AS unitsales, COUNT(*) AS count FROM
stars.transitem AS ti, stars.trans AS t,stars.product as p,
stars.loc AS loc, stars.pgroup AS pg WHERE  ti.transid =
t.transid AND p.pgid = pg.pgid AND t.locid = loc.locid AND
p.prodid = ti.prodid GROUP BY rollup(pg.lineid, (pg.pgid,
pg.pgname)), rollup(loc.country, loc.state),
rollup(year(t.shipdate), month(t.shipdate)) )
```

Check options: [✓ None] Cascaded Local

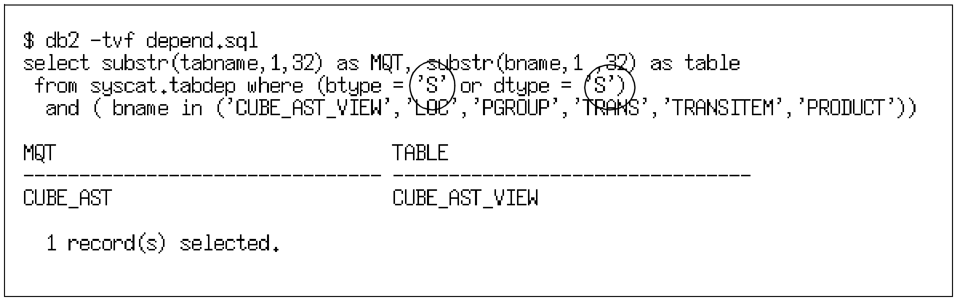
Comment:

Figure 6-37 Definition of cube_ast_view

We identified the underlying tables of the view as being the following:

- stars.loc
- stars.pgroup
- stars.trans
- stars.transitem
- stars.product

Given that we were in a BI workload environment, we investigated the view and the underlying tables for dependencies, as shown in Figure 6-38 on page 446.

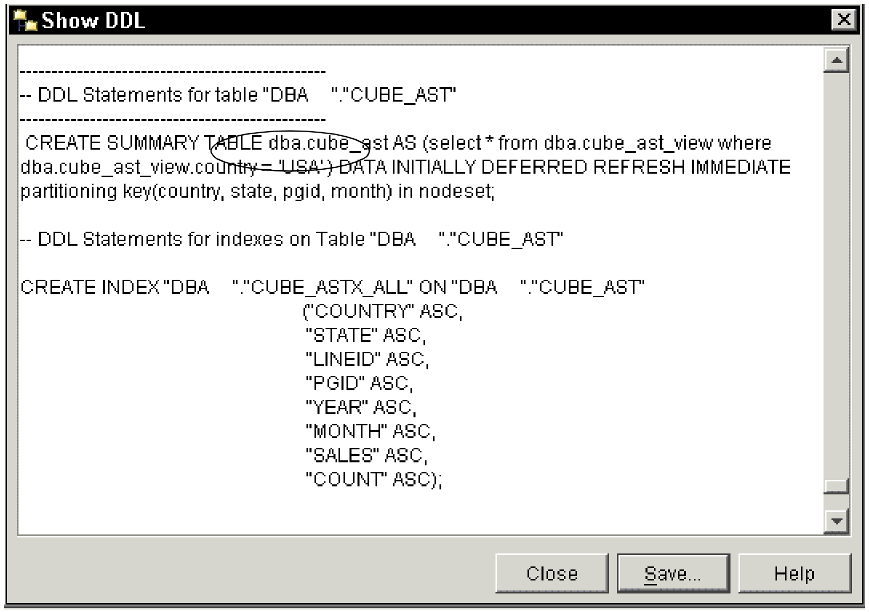


```
$ db2 -tvf depend.sql
select substr(tabname,1,32) as MQT, substr(bname,1,32) as table
 from syscat.tabdep where (btype = 'S' or dtype = 'S')
  and ( bname in ('CUBE_AST_VIEW','LOC','PGROUP','TRANS','TRANSITEM','PRODUCT'))

MQT                              TABLE
-------------------------------- --------------------------------
CUBE_AST                         CUBE_AST_VIEW

  1 record(s) selected.
```

Figure 6-38   Identifying tables dependent on MQTs or Summary Tables

Note: The predicates **`btype = 'S'`** or **`dtype = 'S'`** limit the query results to tables, depending on summary tables or MQTs.

Having identified an MQT on cube_ast_view, we looked up its definition. From the Control Center, we selected the MQT table cube_ast from the tables pane and used the Generate DDL function to view the results shown in Figure 6-39.



```
------------------------------------------------
-- DDL Statements for table "DBA    "."CUBE_AST"
------------------------------------------------
 CREATE SUMMARY TABLE dba.cube_ast AS (select * from dba.cube_ast_view where
dba.cube_ast_view.country = 'USA') DATA INITIALLY DEFERRED REFRESH IMMEDIATE
partitioning key(country, state, pgid, month) in nodeset;

-- DDL Statements for indexes on Table "DBA    "."CUBE_AST"

CREATE INDEX "DBA    "."CUBE_ASTX_ALL" ON "DBA    "."CUBE_AST"
                              ("COUNTRY" ASC,
                               "STATE" ASC,
                               "LINEID" ASC,
                               "PGID" ASC,
                               "YEAR" ASC,
                               "MONTH" ASC,
                               "SALES" ASC,
                               "COUNT" ASC);
```

Figure 6-39   DDL for the cube_ast MQT

A cursory examination shows that the cube_ast MQT has a predicate

```
country = 'USA'
```

that is more restrictive that the original query

```
“select * from dba.cube_ast_view”
```

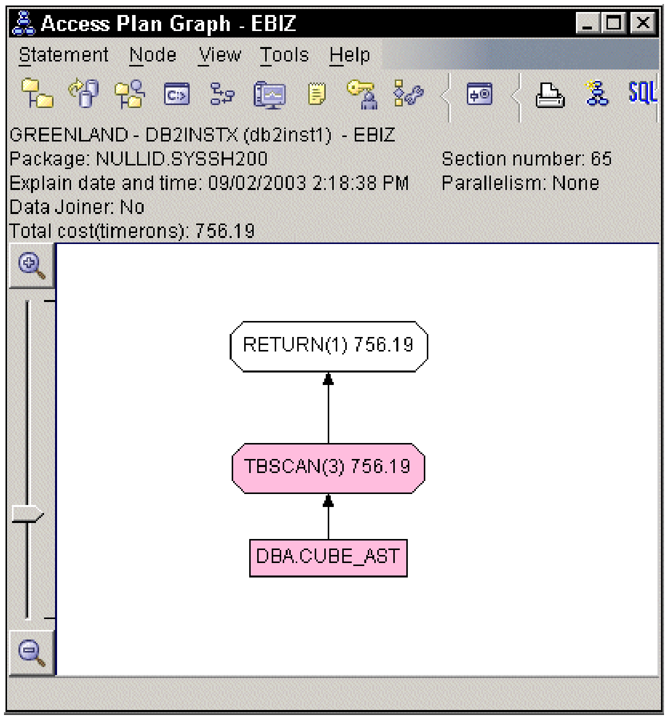
Root cause of the problem

The root cause of the problem is that the query is unable to use the available MQT, because the MQT has a more restrictive predicate defined on it than the original query. The predicates in the MQT *must* be a superset of those specified in the query in order for the DB2 optimizer to make use of the MQT via a query rewrite.

For validation purposes, we modified the query as follows:

```
select * from dba.cube_ast_view where country = “USA”
```

We ran the query through Visual Explain and generated the output shown in Figure 6-40, which shows the query accessing the cube_ast MQT.



```
GREENLAND - DB2INSTX (db2inst1)  - EBIZ
Package: NULLID.SYSSH200                          Section number: 65
Explain date and time: 09/02/2003 2:18:38 PM      Parallelism: None
Data Joiner: No
Total cost(timerons): 756.19
```

Figure 6-40   Visual Explain for revised SQL

Note: This explain does show a table scan being performed, and needs to be tuned to use indexes.

Apply best practices

Keep in mind that designing an MQT requires a thorough understanding of the query workload, and the conditions under which the DB2 optimizer undertakes a query rewrite for it.

While MQTs are powerful tools to achieve superior performance in BI environments, they also have certain overheads such as disk space, locking contention, logging overhead, and response time overheads with certain kinds of refreshes. Refer to 3.3.3, "MQT/AST design considerations" on page 128 for detailed information on MQTs.

6.4 Routine monitoring scenarios

As discussed in "Routine monitoring" on page 5, the objectives of this type of monitoring are to collect information about the workload and stress on the system during periods of normal and peak periods for capacity planning purposes, as well as identifying potential performance problems in the future.

From a problem determination point of view, the objective is to detect deteriorating trends of key performance drivers, and then perform exception monitoring to pinpoint the problem and resolve it before it gets out of hand.

Important: Routine monitoring requires a history repository to determine trends, and reporting mechanisms to alert potential problem conditions via thresholds and alerts.

We did not have such a history repository available, and just asserted the existence of a particular trend for our problem diagnosis purposes.

Here again, adopting a consistent problem determination methodology as described in "Problem determination methodology" on page 7 will ensure speedy resolution of performance problems. However, it should be noted that routine monitoring alerts will enable you to bypass certain validations by virtue of the fact that these alerts point to a specific problem area.

The following subsections cover the following problem determination scenarios:

1. Deteriorating space utilization conditions (BI)
2. Deteriorating buffer pool hit ratios (OLTP)

As in the case of exception events scenarios described in “Exception event scenarios” on page 412, note the following points:

1. The workload and environments were artificially contrived to produce the relevant problem condition for the problem diagnosis exercise; therefore, certain settings can clearly been seen to be “inappropriate” in real world environments.
2. The emphasis of these scenarios is on problem diagnosis, and *not* on problem resolution per se. Best practices for problem resolution are discussed briefly, but not applied to demonstrate the elimination of the problem.

We have similarly organized the description of each routine monitoring scenario as follows:

- Description of the application
- Environment configuration
- Monitor level settings
- Workload used
- Triggering event
- Hypotheses and validations
- Root cause of the problem
- Apply best practices

6.4.1 Deteriorating space utilization conditions (BI)

In this scenario, we investigate how an MDC table with poorly defined dimensions can severely impact space utilization, as well as loading of data. MDCs are primarily used in BI environments.

Description of the application

We used the EBIZ banking application described in Appendix B.3, “EBIZ database” on page 486 as an example of databases used in a BI environment.

Environment configuration

We used a single pSeries Model 650 8 CPU system to run the BI database for this scenario; this workload did not have WebSphere Application Server support.

Table 6-5 on page 450 and Figure 6-41 on page 450 highlight the details of this environment.

Table 6-5 Deteriorating space utilization scenario configuration

Hardware configuration	Software configuration
pSeries Model 650 8 CPUs 32 GB RAM 4 x 146 GB HDD disks	AIX 5.1 ML03 DB2 V8.1 FP1
PC	Windows 2000

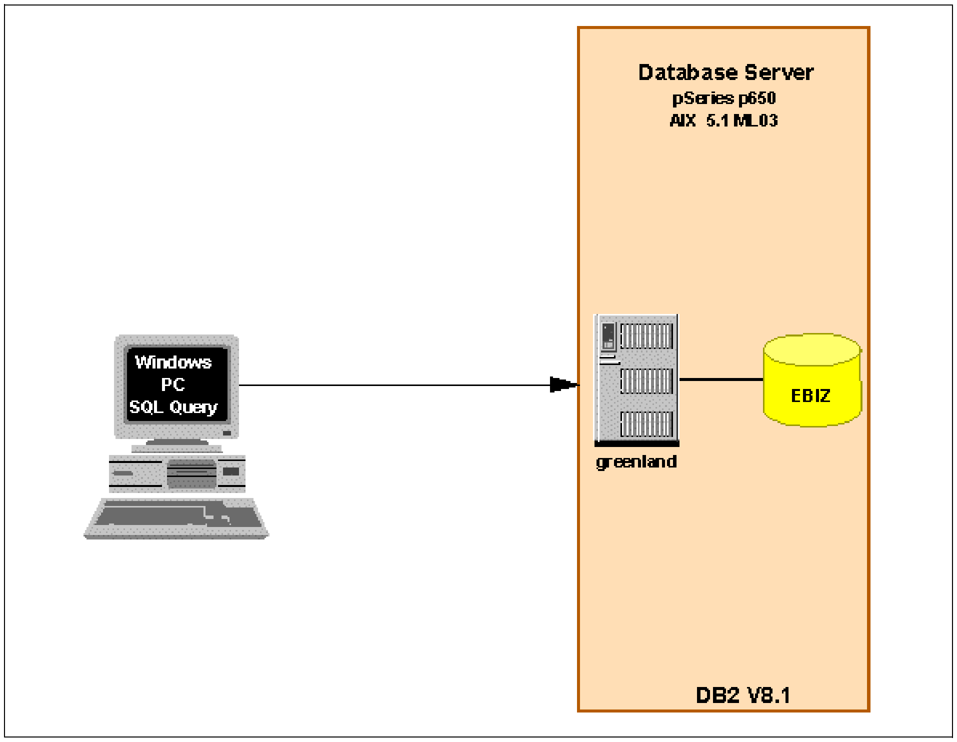


Figure 6-41 Configuration for MDC scenario

Monitor level settings

We used the following recommended settings for routine monitoring in a BI environment as shown in Figure 6-41:

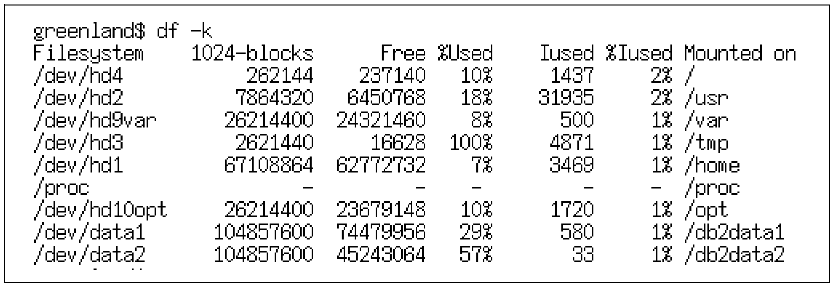
- Set monitor switches **`DFT_MON_BUFPOOL`**, **`DFT_MON_SORT`**, **`DFT_MON_TABLE`**, **`DFT_MON_TIMESTAMP`**, and **`HEALTH_MON`** to ON.
- Set monitor switches **`DFT_MON_LOCK`**, **`DFT_MON_STMT`** and **`DFT_MON_UOW`** to OFF.
- Accept the **`DIAGLEVEL`** default setting (3).

Attention: In addition to gathering snapshot information at appropriate intervals, routine monitoring also involves gathering information about changes in the number and size of files systems, containers, system CPU, memory and I/O utilizations, database resources utilizations and efficiencies such as buffer pool hit ratios, and the application workload itself.

This information should be stored in a repository for subsequent analysis.

Since this scenario is about SMS and DMS table space utilization monitoring, we assumed file system utilization (and number of free blocks) was collected via the **`df`** command for UNIX[1] as shown in Figure 6-42 and the **`db2 list table spaces show details`**[2] command in DB2 as shown in Figure 6-43 on page 452.

Note: Figure 6-42 displays utilization at the filesystem level. To map physical disks to file systems, refer to “Mapping filesystems to physical disks” on page 364.

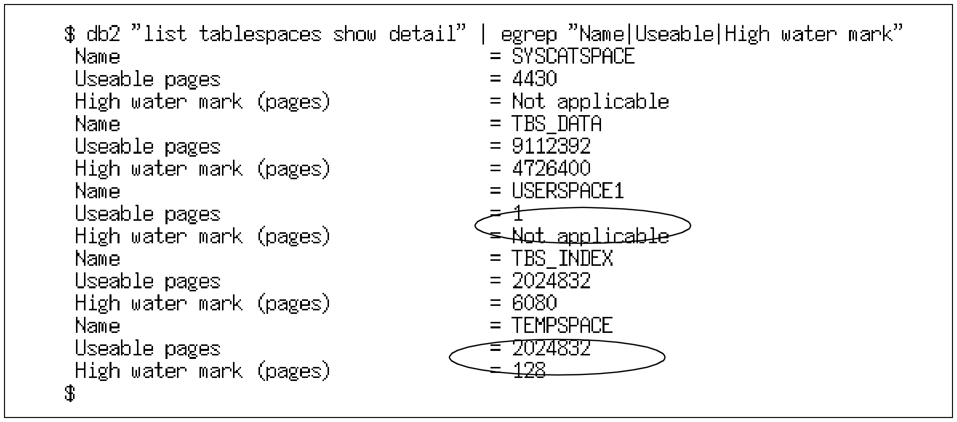


```
greenland$ df -k
Filesystem    1024-blocks      Free %Used    Iused %Iused Mounted on
/dev/hd4           262144    237140   10%     1437     2% /
/dev/hd2          7864320   6450768   18%    31935     2% /usr
/dev/hd9var      26214400  24321460    8%      500     1% /var
/dev/hd3          2621440     16628  100%     4871     1% /tmp
/dev/hd1         67108864  62772732    7%     3469     1% /home
/proc                   -         -     -        -     -  /proc
/dev/hd10opt     26214400  23679148   10%     1720     1% /opt
/dev/data1      104857600  74479956   29%      580     1% /db2data1
/dev/data2      104857600  45243064   57%       33     1% /db2data2
```

Figure 6-42 df command showing filesystem utilization

[1] For Windows environments, in Windows Explorer, right-click the volume and select **Properties**.

[2] In a partitioned database environment, this command does not return all the tablespaces in the database. To obtain a list of all the tablespaces, query SYSCAT.SYSTABLESPACES.



```
$ db2 "list tablespaces show detail" | egrep "Name|Useable|High water mark"
 Name                                = SYSCATSPACE
 Useable pages                       = 4430
 High water mark (pages)             = Not applicable
 Name                                = TBS_DATA
 Useable pages                       = 9112392
 High water mark (pages)             = 4726400
 Name                                = USERSPACE1
 Useable pages                       = 1
 High water mark (pages)             = Not applicable
 Name                                = TBS_INDEX
 Useable pages                       = 2024832
 High water mark (pages)             = 6080
 Name                                = TEMPSPACE
 Useable pages                       = 2024832
 High water mark (pages)             = 128
$
```

Figure 6-43 Finding DMS tablespace high water marks

Note: The high water mark value is only applicable for DMS tablespaces (and not SMS tablespaces), as shown in Figure 6-43.

Workload used

The workload consisted of several BI SQL queries against the EBIZ database.

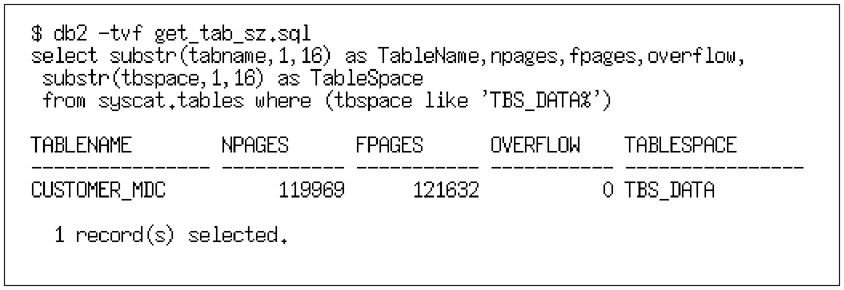
Triggering event

Analysis of tablespace utilization collected through routine monitoring indicated that the DMS tablespace TBS_DATA was unexpectedly consuming disk space at a rate that would soon require providing additional containers.

Since the growth rate exceeded anticipated growth rates, it merited further investigation.

Hypotheses and validations

We needed to determine the database objects in this tablespace in order to pinpoint the particular object causing the unexpected growth. Figure 6-44 on page 453 indicated that there was only one table **`CUSTOMER_MDC`** in this table space.

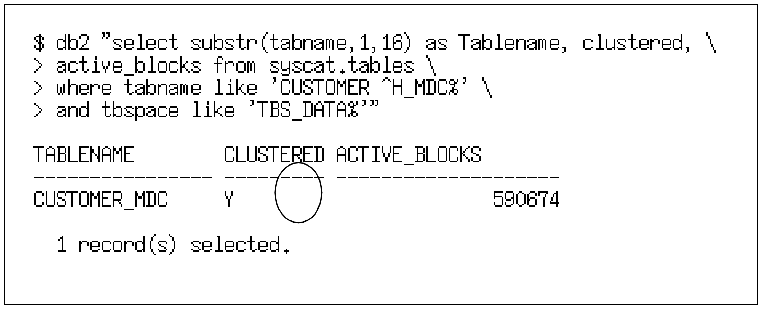


```
$ db2 -tvf get_tab_sz.sql
select substr(tabname,1,16) as TableName,npages,fpages,overflow,
 substr(tbspace,1,16) as TableSpace
 from syscat.tables where (tbspace like 'TBS_DATA%')

TABLENAME        NPAGES      FPAGES      OVERFLOW    TABLESPACE
---------------- ----------- ----------- ----------- ----------------
CUSTOMER_MDC          119969      121632           0 TBS_DATA

  1 record(s) selected.
```

Figure 6-44   Identifying table space usage in a table space

Note: Had there been more than one table identified, we may have needed to monitor the growth of relevant tables over time to identify the high growth table.

A lookup of SYSCAT.SYSTABLES confirmed that the table **`CUSTOMER_MDC`** is an MDC table (indicated by `Y` in the CLUSTERED column).

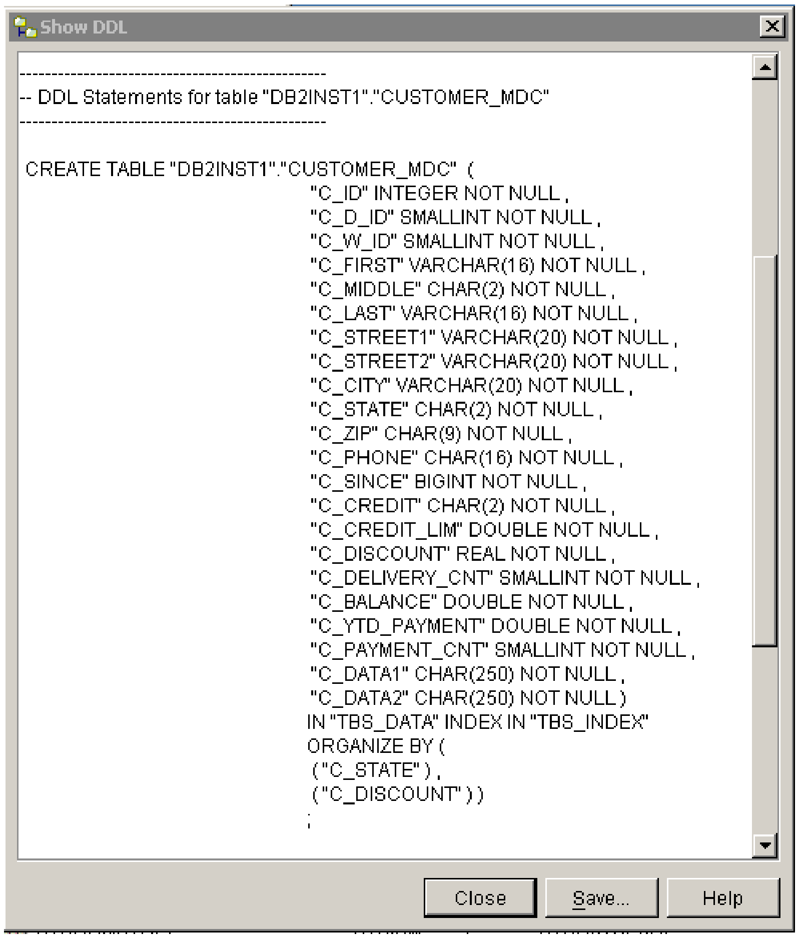


```
$ db2 "select substr(tabname,1,16) as Tablename, clustered, \
> active_blocks from syscat.tables \
> where tabname like 'CUSTOMER ^H_MDC%' \
> and tbspace like 'TBS_DATA%'"

TABLENAME        CLUSTERED ACTIVE_BLOCKS
---------------- --------- --------------------
CUSTOMER_MDC     Y                       590674

  1 record(s) selected.
```

Figure 6-45   Identifying an MDC table

Since we were aware of the space utilization issues of poorly chosen dimension keys, we decided to compute the cell count for the dimensions of this table.

We identified the dimension columns of this table to be C_STATE and C_DISCOUNT by using the Control Center, selecting the **CUSTOMER_MDC** table, then selecting **Generate SQ**L. The output is shown in Figure 6-46 on page 454, and the number of rows in this table as shown in Figure 6-47 on page 454.



```
------------------------------------------------
-- DDL Statements for table "DB2INST1"."CUSTOMER_MDC"
------------------------------------------------

 CREATE TABLE "DB2INST1"."CUSTOMER_MDC"  (
                  "C_ID" INTEGER NOT NULL ,
                  "C_D_ID" SMALLINT NOT NULL ,
                  "C_W_ID" SMALLINT NOT NULL ,
                  "C_FIRST" VARCHAR(16) NOT NULL ,
                  "C_MIDDLE" CHAR(2) NOT NULL ,
                  "C_LAST" VARCHAR(16) NOT NULL ,
                  "C_STREET1" VARCHAR(20) NOT NULL ,
                  "C_STREET2" VARCHAR(20) NOT NULL ,
                  "C_CITY" VARCHAR(20) NOT NULL ,
                  "C_STATE" CHAR(2) NOT NULL ,
                  "C_ZIP" CHAR(9) NOT NULL ,
                  "C_PHONE" CHAR(16) NOT NULL ,
                  "C_SINCE" BIGINT NOT NULL ,
                  "C_CREDIT" CHAR(2) NOT NULL ,
                  "C_CREDIT_LIM" DOUBLE NOT NULL ,
                  "C_DISCOUNT" REAL NOT NULL ,
                  "C_DELIVERY_CNT" SMALLINT NOT NULL ,
                  "C_BALANCE" DOUBLE NOT NULL ,
                  "C_YTD_PAYMENT" DOUBLE NOT NULL ,
                  "C_PAYMENT_CNT" SMALLINT NOT NULL ,
                  "C_DATA1" CHAR(250) NOT NULL ,
                  "C_DATA2" CHAR(250) NOT NULL )
                 IN "TBS_DATA" INDEX IN "TBS_INDEX"
                 ORGANIZE BY (
                  ("C_STATE") ,
                  ("C_DISCOUNT"))
                 ;
```

Figure 6-46 Identifying dimension columns of CUSTOMER_MDC table

```
$ db2 "select count(*) from customer_mdc"

1
-----------
     600000

  1 record(s) selected.
```

Figure 6-47 Number of rows in CUSTOMER_MDC table

We determined the cell count for the CUSTOMER_MDC table as shown in Figure 6-48 on page 455, which indicated a cell count (590674) almost equal to the number of rows in the table (600000). This would cause an extent to be used to store a single row for each distinct two-column pair.

```
$ db2 -tvf cell_check.sql
with cell_table as (
 select distinct c_state, c_discount from customer_mdc)
 select count(*) as cell_count from cell_table

CELL_COUNT
-----------
     590674

  1 record(s) selected.
```

Figure 6-48 Output from cell count table

To verify that the chosen dimension columns had been poorly designed, we computed the Rows per Cell (RpC) for these dimensions as shown in Figure 6-49.

```
$ db2 -tvf rpc.sql
with cell_table ( c_state, c_discount, RpC ) as (
 select distinct c_state, c_discount, count(*) from customer_mdc
  group by c_state,c_discount )
 select avg(RpC) as RpC, min(Rpc) as MinRpC,
  max(RpC) as MaxRpC from cell_table

RPC         MINRPC      MAXRPC
----------- ----------- -----------
          1           1           3

  1 record(s) selected.
```

Figure 6-49 Query showing RpC count for candidate dimensions

The low number of rows in each cell confirms that C_STATE and C_DISCOUNT are a poor choice for the dimension columns, since the size of each cell is at least one block (extent). RpC indicates that only 1 to 3 rows per cell are being stored in a block.

Root cause of the problem

The root cause of unexpected space utilization is a poor selection of dimension columns for an MDC table, which resulted in many extents being created with very few rows in each one.

Apply best practices

MDC table space utilization can be improved by changing the number of cells, the size of a cell's allocation unit (block = extent), or both. Specifically, you can:

- Change the extent size
- Change the page size
- Change the granularity of one or more dimensions
- Change the number candidate dimensions
- Try a different combination of dimensions

Note: Since each of these changes requires the table to be dropped and recreated, great care must be exercised in the design of MDC tables to avoid excessive space utilization or data load problems, in order to avoid these drastic measures.

6.4.2 Deteriorating buffer pool hit ratios (OLTP)

In this scenario, we highlight the benefits of high buffer pool hit ratios, the point of diminishing returns with buffer pool sizes, and the overall impact of buffer pool paging on end user response times.

Hit ratios are computed as

Hit ratio = (1 - #physical reads/#logical reads)*100

High hit ratios (90%-95% for OLTP environments and 85% for BI environments) are desirable for minimizing I/O and improving overall end user response times.

The buffer pool hit ratio can often be improved by increasing the size of the buffer pools, but this depends upon many factors including the size of the tables and indexes involved, sequential or random access characteristics of the application workload, and the contention amongst tables and indexes sharing the same buffer pool.

Description of the application

We used the Trade2 application described in Appendix B.4, “Trade2 database and application” on page 487 as the OLTP environment.

Environment configuration

We used a single AIX pSeries Model 270 to run the DB2 database, and a single IBM eServer xSeries® 330 to run WebSphere Application Server. A Windows 2000 PC used the akstress tool to drive the Trade2 application.

Table 6-6 on page 457 and Figure 6-50 on page 457 highlight the details of the configuration.

Table 6-6 Buffer pool hit ratio scenario configuration

Hardware configuration	**Software configuration**
IBM eServer xSeries 330 2 CPUs 1.1 GHz each 4 GB memory 36 GB disk	Windows 2000 Fixpak 2 WebSphere Application Server 4.0.5
pSeries Model 270 2 CPUs 1GB RAM 2 x 36 GB SCSI disks	AIX 5.1 ML03 DB2 V8.1 FP1
PC	Windows 2000 akstress tool

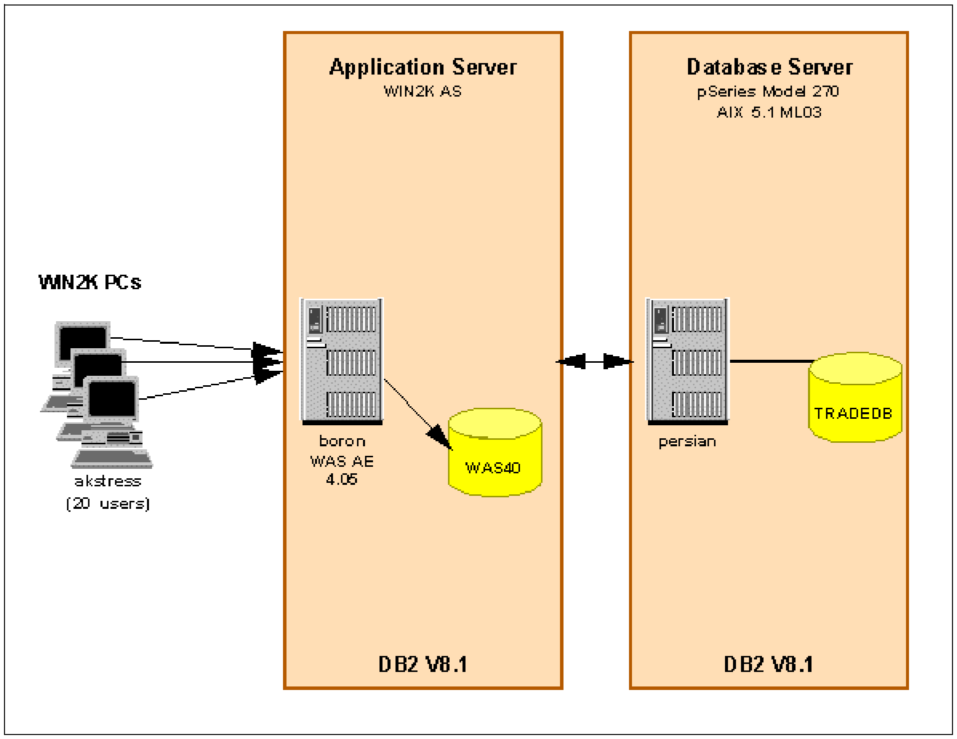


Figure 6-50 Buffer pool hit ratio scenario configuration

Attention: We used the `rmss` command to adjust the apparent size of the memory on the pSeries 270 system to demonstrate this particular scenario. This command locks a specified amount of memory so that it cannot be used by applications, and effectively simulates a system with less memory.

We configured the system with 632 MB by using the command:

```
rmss -c 632
```

This value was chosen after some trial and error to enable us to best demonstrate the scenario.

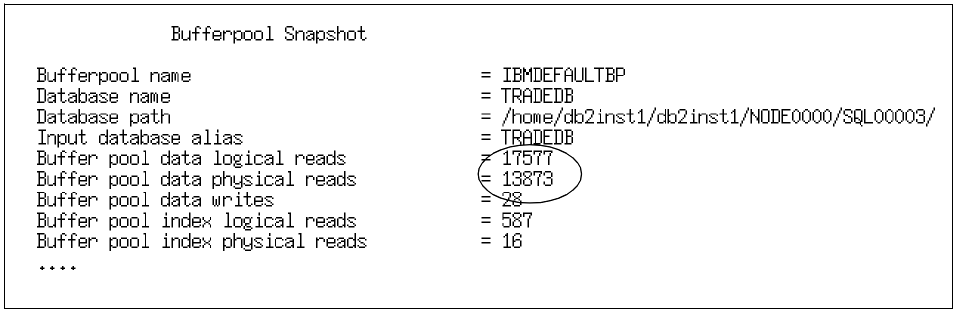
Monitor level settings

We used the following recommended settings for routine monitoring in an OLTP environment, as shown in Figure 6-6 on page 415:

- Set monitor switches **`DFT_MON_BUFPOOL`**, **`DFT_MON_SORT`**, **`DFT_MON_TABLE`**, **`DFT_MON_TIMESTAMP`**, **`DFT_MON_UOW`**, and **`HEALTH_MON`** to ON.
- Set monitor switches **`DFT_MON_LOCK`** and **`DFT_MON_STMT`** to OFF.
- Accept the **`DIAGLEVEL`** default setting (3).

As part of the routine monitoring process, information is gathered about buffer pool efficiencies using snapshot, and then written to a history repository for further analysis.

Since this scenario is about buffer pool hit ratios, buffer pool information shown in Figure 6-51 was collected for each buffer pool and trends analyzed over time.



```
             Bufferpool Snapshot

Bufferpool name                           = IBMDEFAULTBP
Database name                             = TRADEDB
Database path                             = /home/db2inst1/db2inst1/NODE0000/SQL00003/
Input database alias                      = TRADEDB
Buffer pool data logical reads            = 17577
Buffer pool data physical reads           = 13873
Buffer pool data writes                   = 28
Buffer pool index logical reads           = 587
Buffer pool index physical reads          = 16
....
```

Figure 6-51 Buffer pool snapshot

Workload used

We used the Trade2 interactive trading application and the akstress tool to simulate 20 concurrent users, each performing a login followed by several queries and a logout. We drove this workload for 20 minutes.

Triggering event

Analysis of buffer hit ratios collected through routine monitoring indicated that the IBMDEFAULTBP buffer pool hit ratio had deteriorated suddenly, with the latest measurements showing an alarming buffer pool hit ratio of 21%.

Hypotheses and validations

The reasons for the sudden drop in buffer pool hit ratios could be due to many factors, including the following:

- A sudden increase in the number of applications assigned to the buffer pool.

  We used the **`db2 list applications`** command to view the number of connected users.

- A sudden growth in the size of the tables associated with the buffer pool. This could be determined by comparison with historical data.

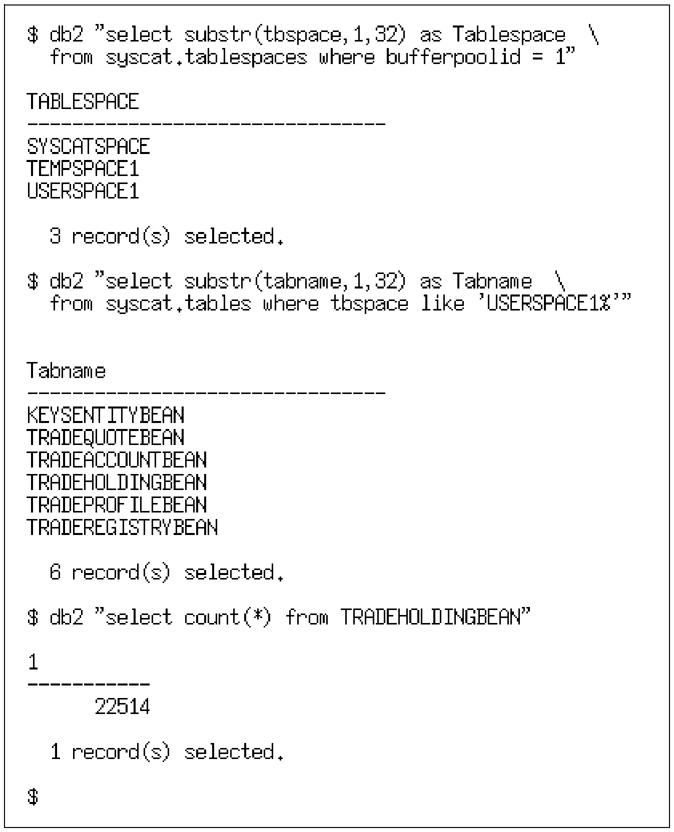
  The default buffer pool, IBMDEFAULTBP, has a “**`bufferpoolid`**” of 1.

  As shown in Figure 6-52 on page 460, we checked the number and names of tablespaces using the default buffer pool. We ignored the system catalogue and temporary tablespaces and identified the names of tables using the “data” tablespace, USERSPACE1. We then looked at the row count of each table; only one is shown. We observed that TRADEHOLDINGBEAN was increasing in size compared to historical data.

- A change in the workload profile resulting in more random access to data.

  This hypothesis is not easy to validate. It may be necessary to consult application developers and application users to determine whether either group have changed their work patterns.

  The snapshot variable **`Unread prefetch pages`**, available in the BufferPool Snapshot can provide evidence. This variable counts the number of pages that the prefetcher read that were never used. An increase in this variable compared to previous values indicates increased random access to data.



```
$ db2 "select substr(tbspace,1,32) as Tablespace  \
  from syscat.tablespaces where bufferpoolid = 1"

TABLESPACE
--------------------------------
SYSCATSPACE
TEMPSPACE1
USERSPACE1

  3 record(s) selected.

$ db2 "select substr(tabname,1,32) as Tabname  \
  from syscat.tables where tbspace like 'USERSPACE1%'"


Tabname
--------------------------------
KEYSENTITYBEAN
TRADEQUOTEBEAN
TRADEACCOUNTBEAN
TRADEHOLDINGBEAN
TRADEPROFILEBEAN
TRADEREGISTRYBEAN

  6 record(s) selected.

$ db2 "select count(*) from TRADEHOLDINGBEAN"

1
-----------
      22514

  1 record(s) selected.

$
```

Figure 6-52   Determining sizes of tables using a specific buffer pool

Root cause of the problem

Investigations showed a combination small default buffer pool size in conjunction with an increase in the number of applications being assigned to the buffer pool causing contention, thereby negatively affecting the efficiency of the buffer pool.

Apply best practices

Tuning the buffer pool is a trial and error process.

We gradually increased the buffer pool in our contrived environment (from 32 pages to 64 pages to 128 pages to 256 pages to 512 pages to 1024 pages), measuring at each point the buffer pool hit ratios, as well as the end-user response times that we obtained from the akstress tool.

The results of these tests are shown in Figure 6-53 on page 461.

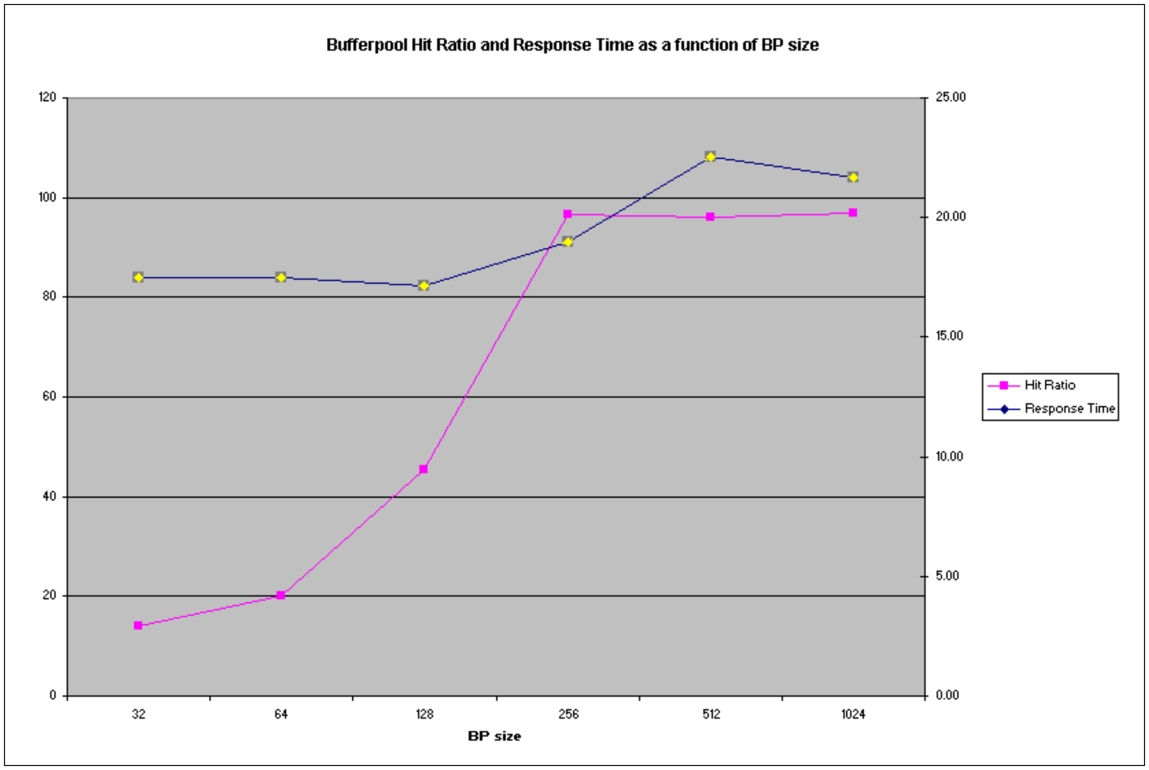


Figure 6-53 Buffer pool hit ratio & response time as function of buffer pool size

Figure 6-53 shows a hit ratio of 95% with a buffer pool size of 256 pages, with a flattening out even as the number of buffers were increased. The point of diminishing returns is somewhere between 256 pages and 512 pages, and you could decrease the step increase in buffer pool size to get to a more accurate point.

Important: The more interesting metric is the end-user response time, which shows deterioration occurring from the 128 buffers configuration level even as buffer hit ratios continued to rise.

This is due to memory paging of the buffer pool, which effectively marks the “real” point of diminishing returns somewhere between the 128 buffers and 256 buffers configuration.

Therefore, increasing buffer pool hit ratios are not the only metric to determine the efficiency of a buffer pool; we also need to take into account the impact of increasing the size of the buffer pool on system paging and application response times.

The “real” point of diminishing returns is the proper balance between buffer pool hit ratios, application response times, and acceptable system paging.

6.5 Online/Realtime monitoring scenarios

As discussed in “Routine monitoring” on page 5, the objective of this type of monitoring is to be on the lookout for specific events that may either identify a specific problem, or portend problems in the near to immediate future, in order to take prompt corrective action. “Near to immediate future” implies minutes, rather than hours.

The key to this type of monitoring is that it involves looking for specific events in a short interval of time (short history) that are known to degrade performance, and having the option to take prompt corrective action to rectify the problem. In other words, there probably needs to be a very short delay between information collection and a corrective response. One example of such an event is the occurrence of an excessive number of deadlocks in a short period of time, which need to be addressed promptly to ensure that business objectives are not being compromised.

Note: The need to minimize the overhead of online/realtime monitoring is critical, given that most problems manifest themselves at peak loads.

Here again, adopting a consistent problem determination methodology as described in “Problem determination methodology” on page 7 will ensure speedy resolution of performance problems. However, it should be noted that online/realtime monitoring is similar to routine monitoring, since its alerts will enable you to bypass certain validations by virtue of the fact that these alerts point to a specific problem area.

The Health Monitor and Health Center are the primary tools for online/realtime monitoring in DB2.

In this section, we will cover a lock contention scenario in an OLTP environment.

As in the case of exception events scenarios described in “Exception event scenarios” on page 412, note the following points:

1. The workload and environments were artificially contrived to produce the relevant problem condition for the problem diagnosis exercise; therefore,

certain settings can clearly been seen to be “inappropriate” in real world environments.

2. The emphasis of these scenarios is on problem diagnosis, and *not* on problem resolution per se. Best practices for problem resolution are discussed briefly, but not applied to demonstrate the elimination of the problem.

We have similarly organized the description of the lock contention scenario as follows:

- Description of the application
- Environment configuration
- Monitor level settings
- Workload used
- Triggering event
- Hypotheses and validations
- Root cause of the problem
- Apply best practices

6.5.1 Lock contention (OLTP)

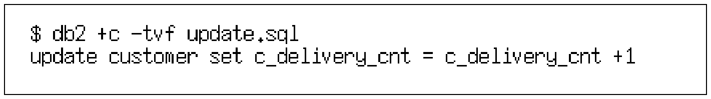
Lock contention is a common and natural occurrence in any multiuser application environment. However, to achieve better throughput and faster response times, applications must be designed to minimize lock contention and the DBA must ensure that the database configuration parameters LOCKLIST and MAXLOCKS are configured appropriately.

In this scenario, we monitor application concurrency health indicators by setting warning/alarm thresholds on lock escalation rate, lock list utilization, deadlock rate, and percentage of applications waiting on locks. We then pinpoint the cause of the problem.

Description of the application

We used the DTW application described in Appendix B.2, “DTW workload” on page 486, which is a slightly modified version of the industry standard TPC-C OLTP benchmark, and an uncommitted SQL statement to simulate contending applications to demonstrate this scenario.

Figure 6-54 on page 464 shows a batch file consisting of an SQL UPDATE statement that does not commit. This will result in all updated rows being locked until a commit is issued.



```
$ db2 +c -tvf update.sql
update customer set c_delivery_cnt = c_delivery_cnt +1
```

Figure 6-54 Uncommitted SQL update statement

Note: Our intention with the SQL update in Figure 6-54 was to simulate the problem of applications performing “infrequent commits”, which can undermine concurrency and result in erratic response times.

Environment configuration

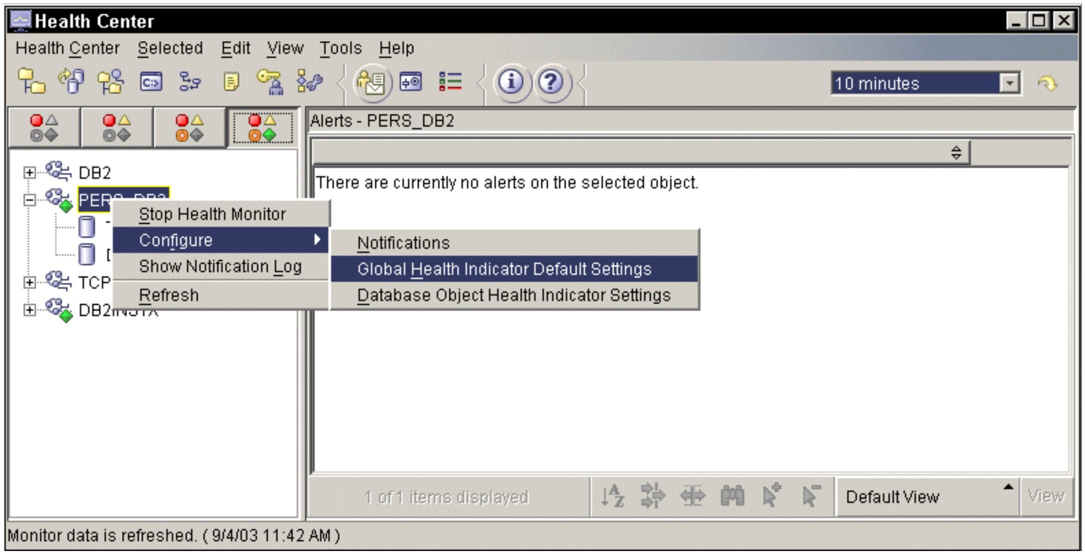
We used the same system configuration shown in Table 6-3 on page 427 and Figure 6-17 on page 427.

Monitor level settings

Since this is an OLTP environment, we used the following recommended settings as shown in Figure 6-6 on page 415:

- Set monitor switches **DFT_MON_BUFPOOL**, **DFT_MON_SORT**, **DFT_MON_TABLE**, **DFT_MON_TIMESTAMP**, **DFT_MON_UOW**, and **HEALTH_MON** to ON.
- Set monitor switches **DFT_MON_LOCK** and **DFT_MON_STMT** to OFF.
- Accept the **DIAGLEVEL** default setting (3).

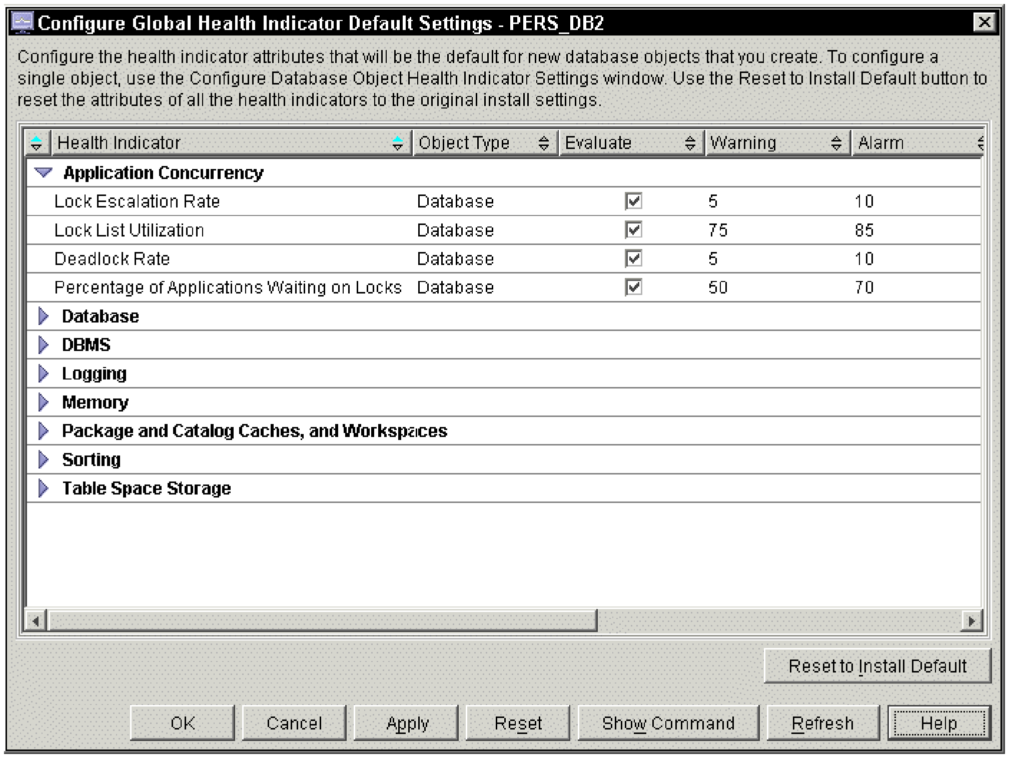
We used the Health Center to configure Global Health Indicators on the DB2 instance PERS_DB2, as shown in Figure 6-55 on page 465, to provide an alert if application concurrency-related health indicators exceeded certain thresholds, as shown in Figure 6-56 on page 466.



There are currently no alerts on the selected object.

Monitor data is refreshed. (9/4/03 11:42 AM)

Figure 6-55 Configuring Health Monitor Global Health Indicators



Configure Global Health Indicator Default Settings - PERS_DB2

Configure the health indicator attributes that will be the default for new database objects that you create. To configure a single object, use the Configure Database Object Health Indicator Settings window. Use the Reset to Install Default button to reset the attributes of all the health indicators to the original install settings.

Health Indicator	Object Type	Evaluate	Warning	Alarm
Application Concurrency				
Lock Escalation Rate	Database	✓	5	10
Lock List Utilization	Database	✓	75	85
Deadlock Rate	Database	✓	5	10
Percentage of Applications Waiting on Locks	Database	✓	50	70
Database				
DBMS				
Logging				
Memory				
Package and Catalog Caches, and Workspaces				
Sorting				
Table Space Storage				

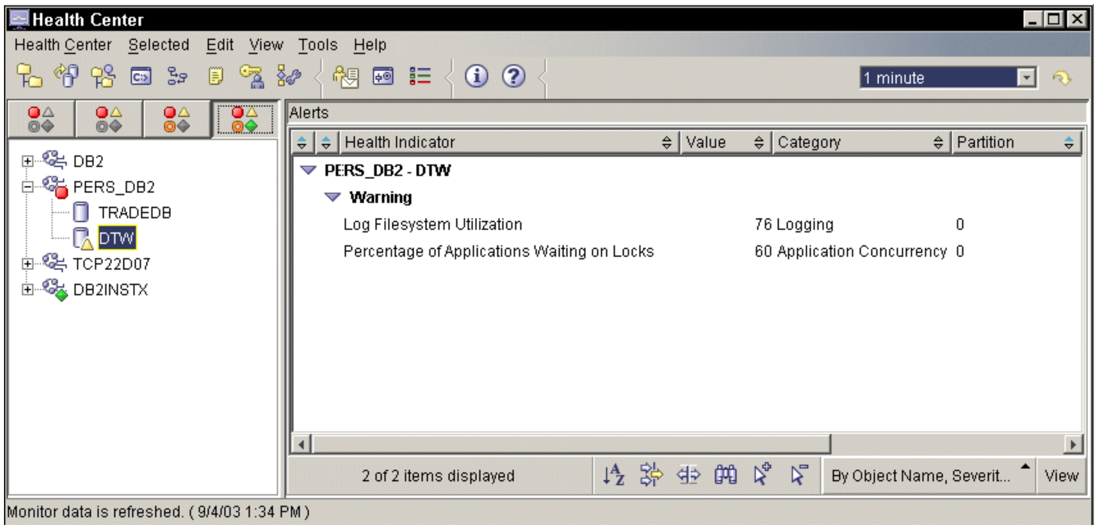
Figure 6-56 Health Monitor - setting lock alerts

Workload used

We used several DTW driver programs to perform a variety of queries, updates, inserts, and deletes against the DTW database. This provided the background OLTP workload, and we ran our uncommitted SQL update statement shown in Figure 6-54 on page 464 to create lock contention problems.

Triggering event

An alert from the Health Center that the number of applications waiting on locks exceeded the configured threshold as shown in Figure 6-57 on page 467.

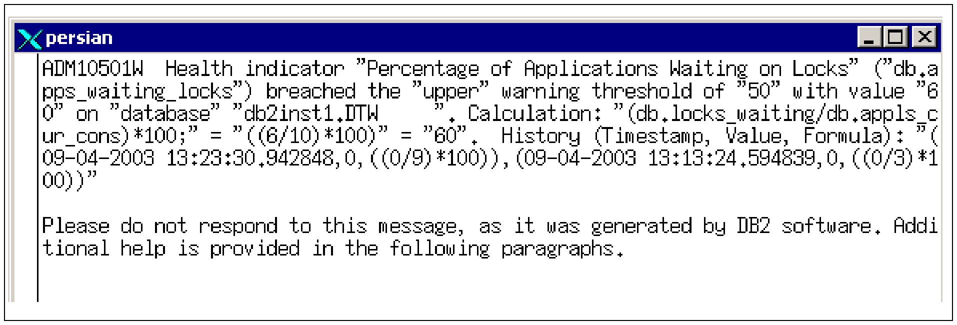


```
PERS_DB2 - DTW
  Warning
    Log Filesystem Utilization                    76 Logging                 0
    Percentage of Applications Waiting on Locks   60 Application Concurrency 0
```

Monitor data is refreshed. (9/4/03 1:34 PM)

Figure 6-57 Health Monitor Alert

We also received an e-mail about the alert, as shown in Figure 6-58.

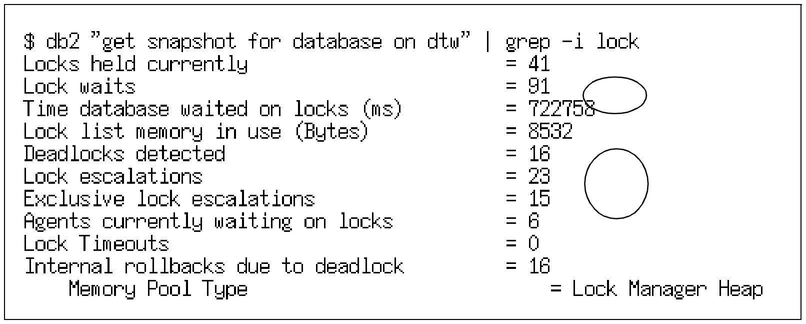


```
ADM10501W  Health indicator "Percentage of Applications Waiting on Locks" ("db.a
pps_waiting_locks") breached the "upper" warning threshold of "50" with value "6
0" on "database" "db2inst1.DTW     ". Calculation: "(db.locks_waiting/db.appls_c
ur_cons)*100;" = "((6/10)*100)" = "60". History (Timestamp, Value, Formula): "(
09-04-2003 13:23:30.942848,0,((0/9)*100)),(09-04-2003 13:13:24.594839,0,((0/3)*1
00))"

Please do not respond to this message, as it was generated by DB2 software. Addi
tional help is provided in the following paragraphs.
```

Figure 6-58 Alert e-mail

Hypotheses and validations

The Health Center alert enables us to focus directly on the locking considerations, and we gathered snapshot information as shown in Figure 6-59 on page 468.



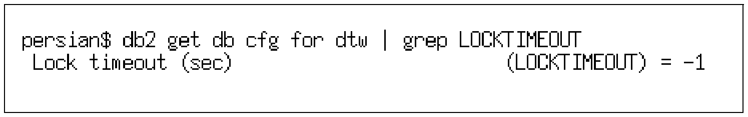
```
$ db2 "get snapshot for database on dtw" | grep -i lock
Locks held currently                       = 41
Lock waits                                 = 91
Time database waited on locks (ms)         = 722758
Lock list memory in use (Bytes)            = 8532
Deadlocks detected                         = 16
Lock escalations                           = 23
Exclusive lock escalations                 = 15
Agents currently waiting on locks          = 6
Lock Timeouts                              = 0
Internal rollbacks due to deadlock         = 16
    Memory Pool Type                           = Lock Manager Heap
```

Figure 6-59 Lock information from database snapshot

There appear to be a number of problems besides the lock wait alert received as follows:

- 91 `Lock waits` with an average lock wait time of (722758 / 91) = 7.9 seconds
- 23 `Lock escalations`, with 15 `Exclusive lock escalations`
- 16 `Deadlocks detected`
- 0 `Lock timeouts`
- 16 `Internal rollbacks due to deadlock`

The absence of lock timeouts caused us to check the LOCKTIMEOUT database configuration parameter, as shown in Figure 6-60.

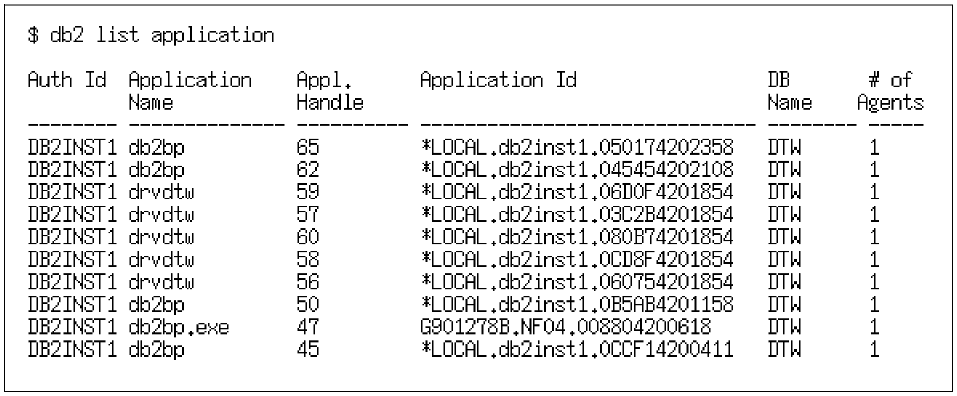


```
persian$ db2 get db cfg for dtw | grep LOCKTIMEOUT
 Lock timeout (sec)                          (LOCKTIMEOUT) = -1
```

Figure 6-60 Current value of LOCKTIMEOUT

The -1 setting explains the lack of lock timeouts, but it does not explain the reason for the lock escalations and deadlocks.

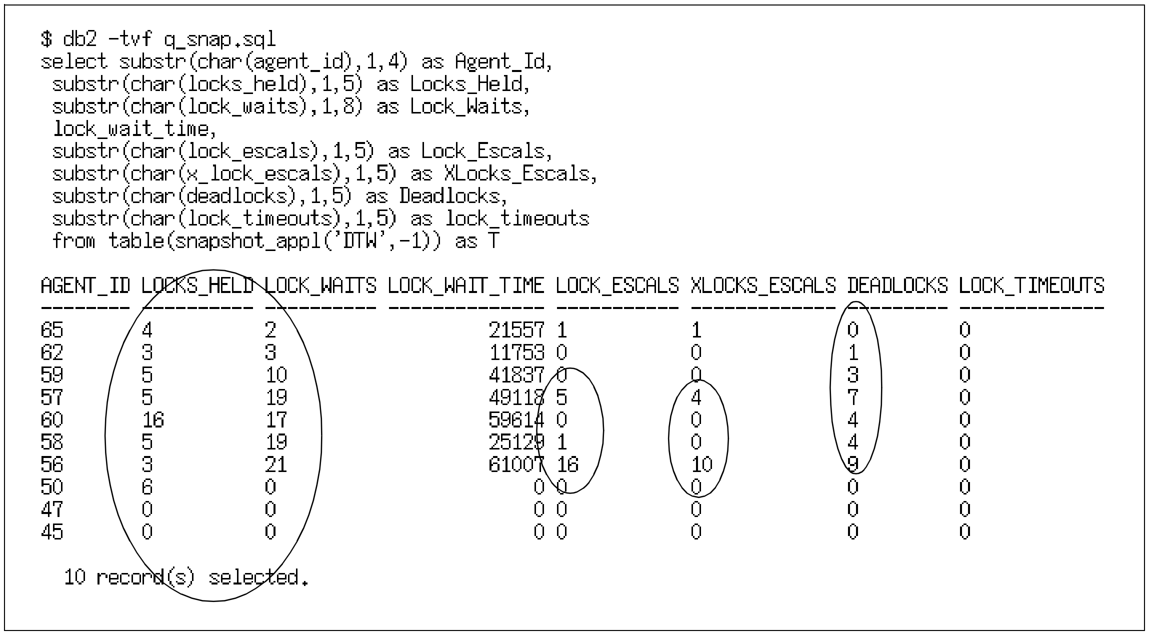
We issued a **`db2 list applications`** command to review current applications, as shown in Figure 6-61 on page 469.



```
$ db2 list application

Auth Id  Application    Appl.      Application Id                 DB       # of
         Name           Handle                                    Name     Agents
-------- -------------- ---------- ------------------------------ -------- -----
DB2INST1 db2bp          65         *LOCAL.db2inst1.050174202358   DTW      1
DB2INST1 db2bp          62         *LOCAL.db2inst1.045454202108   DTW      1
DB2INST1 drvdtw         59         *LOCAL.db2inst1.06D0F4201854   DTW      1
DB2INST1 drvdtw         57         *LOCAL.db2inst1.03C2B4201854   DTW      1
DB2INST1 drvdtw         60         *LOCAL.db2inst1.080B74201854   DTW      1
DB2INST1 drvdtw         58         *LOCAL.db2inst1.0CD8F4201854   DTW      1
DB2INST1 drvdtw         56         *LOCAL.db2inst1.060754201854   DTW      1
DB2INST1 db2bp          50         *LOCAL.db2inst1.0B5AB4201158   DTW      1
DB2INST1 db2bp.exe      47         G901278B.NF04.008804200618     DTW      1
DB2INST1 db2bp          45         *LOCAL.db2inst1.0CCF14200411   DTW      1
```

Figure 6-61 Applications running

In DB2 V8.1, snapshot data is written to system tables that may be accessed via **get snapshot** commands, or using queries against the tables, as shown in Figure 6-62.



```
$ db2 -tvf q_snap.sql
select substr(char(agent_id),1,4) as Agent_Id,
 substr(char(locks_held),1,5) as Locks_Held,
 substr(char(lock_waits),1,8) as Lock_Waits,
 lock_wait_time,
 substr(char(lock_escals),1,5) as Lock_Escals,
 substr(char(x_lock_escals),1,5) as XLocks_Escals,
 substr(char(deadlocks),1,5) as Deadlocks,
 substr(char(lock_timeouts),1,5) as lock_timeouts
 from table(snapshot_appl('DTW',-1)) as T

AGENT_ID LOCKS_HELD LOCK_WAITS LOCK_WAIT_TIME LOCK_ESCALS XLOCKS_ESCALS DEADLOCKS LOCK_TIMEOUTS
-------- ---------- ---------- -------------- ----------- ------------- --------- -------------
65       4          2                   21557 1           1             0         0
62       3          3                   11753 0           0             1         0
59       5          10                  41837 0           0             3         0
57       5          19                  49118 5           4             7         0
60       16         17                  59614 0           0             4         0
58       5          19                  25129 1           0             4         0
56       3          21                  61007 16          10            9         0
50       6          0                       0 0           0             0         0
47       0          0                       0 0           0             0         0
45       0          0                       0 0           0             0         0

  10 record(s) selected.
```

Figure 6-62 Additional lock information from database snapshot

We then examined the db2diag.log and db2inst1.nfy files for additional information. The db2inst1.nfy file provided the most useful information, as shown in Figure 6-63 on page 470.

```
ADM5500W  DB2 is performing lock escalation.  The total number of locks
currently held is "13823", and the target number of locks to hold is "6911".
^^
```

Figure 6-63 db2inst1.nfy file

Lock escalations occur when either the MAXLOCKS or LOCKLIST limits are exceeded. A large number of concurrent applications that do not commit frequently when combined with under configured MAXLOCKS or LOCKLIST can provoke lock escalations.

We determined the LOCKLIST and MAXLOCKS values by issuing the **`db2 get db cfg for dtw`** command, as shown in Figure 6-64.

```
db2 get db cfg for dtw | egrep "LOCKLIST|MAXLOCKS"
 Max storage for lock list (4KB)              (LOCKLIST) = 250
 Percent. of lock lists per application       (MAXLOCKS) = 80
```

Figure 6-64 Current values of MAXLOCKS and LOCKLIST

We attempted to determine which specific limit triggered lock escalation, that is, MAXLOCKS or LOCKLIST.

The current number of locks as listed in the db2inst1.nfy file in Figure 6-63 is 13832. For the sake of the following discussion, let us assume that this was the highest value recorded for all the lock escalations recorded in the db2inst1.nfy file.

We know that in 32 bit DB2, the size of lock structures is 72 bytes for the first lock and 36 bytes for subsequent locks. If we assume a pessimistic average of 60 bytes/lock, then the space required is 60*13832 or 829920 bytes. The size of the locklist is (250 * 4 KB) or 1000KBytes, and this seems to indicate that LOCKLIST was not the cause of the escalation.

Important: Snapshot does not provide a usable high water mark for lock list utilization. However, you can determine the high water mark by using the UNIX command **`db2mtrk -v -d -w`** or by using Memory Visualizer. The **`db2mtrk`** command exists under Windows, but does not provide Lock Manager high water mark information.

It is therefore likely that a single application is exceeding the MAXLOCKS limit, which is 80% or 819200 bytes.

Note: Detecting the applications involved in deadlocks requires exception monitoring such as setting the lock monitor switches and requesting deadlock level monitoring. Given the large number of lock escalations, it seems reasonable to assume that eliminating lock escalations could well resolve the deadlock problem as well. Should that not happen, then exception monitoring for deadlocks should be triggered.

Root cause of the problem

It is most likely that lock escalation is being caused by a single application exceeding the MAXLOCKS limit of 80%, but we cannot be definitive about it.

The most likely root cause of the lock escalations is that an application is trying to obtain more than MAXLOCKS % of the lock list.

Much more detailed analysis needs to be performed to determine which application is exceeding the MAXLOCKS limit, including creating Event Monitors, setting diaglevel to 4, and setting the statement and lock monitor switches.

Apply best practices

It is normally appropriate to increase the MAXLOCKS and LOCKLIST values when lock escalations occur, but in this case, MAXLOCKS is already quite high as compared to the recommendation of 20 to 30% for OLTP environments.

The recommendation here is for the DBA to increase the LOCKLIST value to solve the immediate problem of lock escalations, and then begin a more detailed investigation of errant application programs that are either performing infrequent commits, or updating/inserting/deleting large numbers of rows in a single SQL statement.

A

DB2 UDB ESE Version 8 performance enhancements

In this appendix we briefly highlight new performance enhancements available in DB2 UDB ESE Version 8, and the latest performance enhancements in DB2 UDB Version 8.1.4.

A.1 Introduction

DB2 UDB Version 8 provides major enhancements in availability, manageability, and performance to support the most demanding of OLTP, BI, and mixed workload environments.

In this appendix, we highlight the main performance enhancements in DB2 UDB ESE Version 8 by categorizing them as follows:

- Application-related performance enhancements
- System-related performance enhancements

In the following sections we briefly describe these, as well as the latest performance enhancements in DB2 UDB Version 8.1.4.

Refer to IBM documentation for more detailed information on these enhancements, with particular attention to *DB2 UDB Administration Guide: Performance*, SC09-4821.

A.2 Application-related performance enhancements

These include performance enhancements that primarily impact the application design process under the control of application developers.

We describe these subjects:

- Multidimensional clustering (MDCs)
- Materialized Query Table (MQT) enhancements
- Compression of NULLs and DEFAULT
- LOAD enhancements

A.2.1 Multidimensional clustering

Multidimensional clustering enables a table to be clustered on more than one key (or dimension) simultaneously. DB2 maintains the physical order of data in pages in key order during inserts and deletes using clustering indexes. DB2 also groups pages with the same dimension values into extents.

An MDC table maintains its clustering over all dimensions automatically, and therefore eliminates the need to reorganize the table to restore physical data order. MDCs are indexed using block indexes rather than RID indexes.

MDCs can significantly improve the performance of range queries that have predicates containing one or more keys in the clustering indexes. The DB2 optimizer can apply additional strategies based on the efficiency of the block indexes. In addition, clustering of the data means that only a part of the table needs to be accessed, and this tends to result in more efficient prefetching.

Multidimensional clustering is primarily intended for BI environments, but can also provide performance improvements in OLTP environments.

A.2.2 MQT enhancements

A materialized query table (MQT) is a table whose definition is based on the result of a query, and whose data is derived from data in the tables on which the MQT is based.

Prior to DB2 UDB Version 8, DB2 supported summary tables (automatic summary tables, or ASTs). ASTs are a special type of MQT.

Several features of MQTs improve performance:

- The DB2 optimizer can identify SQL queries that can be satisfied by existing MQTs (ASTs in DB2 UDB Version 7), and rewrite the query to use the MQT instead of the base tables.
- MQTs can now be defined on nicknames, allowing remote data to be cached on the local system. Routing a query to an MQT when all criteria for matching and routing are satisfied results in significant performance improvements as compared to accessing the data remotely.
- MQTs can be “incrementally refreshed”. In this case, a staging table is associated with the MQT. Inserts, deletes and updates to the underlying base tables are immediately reflected in the staging tables. The MQT is refreshed with a `REFRESH TABLE` statement. The staging table is pruned when the refresh is complete.
- DB2 UDB Version V8 also supports the concept of a “user maintained” MQT.

A.2.3 Compression of NULLS and DEFAULT

This is a table property that reduces disk space usage and can consequently increase the performance of large table scans. You can now save disk space for tables that will have many NULLs and SYSTEM DEFAULT values.

Specifying the new options VALUE COMPRESSION and COMPRESS SYSTEM DEFAULT in the CREATE TABLE statement allows null and system default values to be compressed using a new row format.

A.2.4 Load enhancements

This is really more of an availability enhancement. During table loads, the tablespace containing the table is no longer locked, thus allowing users full access to all other tables in the tablespace. If the load is appending data to the table, users have read access to the existing data in the table and to any associated MQTs as well.

A.3 System-related performance enhancements

These include performance enhancements that primarily impact the runtime system performance management activity under the control of system administrators and the DBA.

The following performance enhancements will be described briefly:

- ► Prefetching enhancements
- ► Faster page cleaners
- ► Connection concentrator
- ► Type-2 indexes
- ► Stored procedures and UDFs thread-based model
- ► DMS container enhancements
- ► RUNSTATS enhancements
- ► Logging enhancements
- ► Manageability enhancements

A.3.1 Prefetching enhancements

DB2 UDB Version 8 introduces the concept of block-based buffer pools, which is primarily aimed at improving the performance of prefetching by enabling contiguous pages on disk to be brought into contiguous areas of memory in the buffer pool.

A block-based buffer pool comprises a page area and a block area. The block area consists of blocks where each block contains a number of sequential database pages specified by the **BLOCKSIZE** parameter of the **CREATE** or **ALTER BUFFERPOOL** command.

By default, contiguous pages on disk do not necessarily end up in contiguous pages in the buffer pool. By creating and using block-based buffer pools, sequential database pages will be transferred to a block in the block area of the buffer pool. The prefetch mechanism will read multiple pages into the block area using a single I/O.

Reducing the number of I/Os will improve performance of workloads that perform a great deal of prefetching.

A.3.2 Faster page cleaners

Asynchronous page cleaners write modified buffer pool pages to disk to ensure that there is sufficient free pages in the buffer pool for pages to be read into memory. Configuring the appropriate number of page cleaners has a significant impact on write-intensive databases.

In DB2 UDB Version 8 on AIX systems, the page cleaners use the operating system's asynchronous I/O feature. This reduces the number of page cleaner processes required and improves performance.

Note: AIX's asynchronous I/O feature must be enabled and configured in order for DB2 to take advantage of this performance enhancement.

A.3.3 Connection concentrator

The connection concentrator mechanism is designed to allow many transient connections (such as those from Internet applications) to share a logical coordinating agent. It does this by retaining connection agent processes in a pool after a client connection closes. These agent processes (called *logical agents*) are managed by the coordinating agent, and are reused for subsequent client connections. This results in fewer processes, fewer context switches, and reduced memory usage.

The mechanism is enabled by setting **`MAX_CONNECTIONS`** greater than **`MAX_COORDAGENTS`**.

A.3.4 Type 2 indexes

Type 2 indexes improve performance by eliminating most next-key locking. This is achieved by marking keys as "deleted" rather than actually deleting them. Doing so reduces I/O and improves performance. The deleted keys are physically deleted during periods of low system activity.

Online table reorganization requires Type 2 indexes. Earlier version indexes (now called Type 1 indexes) can be migrated to Type 2 by using the **`REORG INDEXES`** command.

A.3.5 Stored procedures and UDFs thread-based model

In DB2 UDB Version 8, stored procedures, UDFs and methods are implemented using a thread-based model.

Routines that are defined as thread safe run in a single fenced-mode process. There is one process for Java routines and a separate process for non-Java functions. This reduces context switching and also allows resource sharing of the JVM.

Note: In earlier versions of DB2, a separate JVM was created for each routine.

This enhancement results in significant performance gains in environments where a large number of routines run concurrently.

A.3.6 DMS container enhancements

In DB2 UDB Version 8, DMS table spaces provide the following capabilities:

- Drop a container from the tablespace.
- Reduce the size of an existing container.
- Add a new container to the tablespace and control whether or not to rebalance.

Note: The default when adding a container to a DMS tablespace is for a rebalance to occur. This can adversely affect system performance.

A.3.7 RUNSTATS enhancements

The RUNSTATS command has been enhanced to improve the performance of statistics collection and provide additional options such as the SAMPLED option in the DETAILED clause. These include control of distribution limits, faster detailed statistics, and accepting lists of indexes and columns as command parameters.

A.3.8 Logging enhancements

In DB2 UDB Version 8, dual logging is provided on all platforms supported by DB2. The maximum amount of log space that can be defined has increased from 32 GB to 256 GB.

DB2 UDB Version 8 allows an active unit of work to span both active and archive logs. This support for “infinite active logs” can be used to support environments with large jobs that require more log space than you would normally allocate to the primary logs.

A.3.9 Manageability enhancements

While these are not directly performance enhancements, they provide flexibility in managing performance of a DB2 environment.

The include the following:

- **Online configurable parameters**

  A significant number of database manager and database configuration parameters can now be altered online without disconnecting users from the database. Current values, as well as the value to be used on next start, can be viewed using the **`SHOW DETAILS`** option to the **`GET DATABASE`** and **`GET DATABASE MANAGER CONFIGURATION`** commands.

- **Online buffer pool changes**

  In addition to various online configurable manager and database configuration parameters, buffer pools can now be created, altered, and dropped, without stopping the database.

- **Configuration Advisor and AUTOCONFIGURE command**

  A Configuration Advisor Wizard and an **`autoconfigure`** command are provided to help tune key DB2 database manager and database configuration parameters to the unique requirements of your environment. The suggested configuration parameters may be set immediately, or you can have the wizard/command create a script of suggested parameter settings for review and later execution.

- **Health Monitor and Health Center**

  These two new features help monitor the health of the DB2 system. DB2 also provides a set of health indicators to evaluate specific aspects of database manager or database performance.

- **Memory Visualizer**

  Memory Visualizer is new GUI interface for uncovering and fixing memory-related problems of an instance. Memory Visualizer may be invoked from Health Center recommendations, or independently, as its own monitoring tool from the Control Center.

A.4 DB2 UDB Version 8.1.4

Fixpak 4 became available November 2003, and it supports a number of performance enhancements, including the following:

- Backup compression
- Direct I/O support on AIX
- Searched statements against fullselects
- Result set retrieval from SQL data-change operations
- Range-clustered tables
- Asymmetric index splitting
- Temporary tables in SMS
- Page cleaning enhancements
- Monitoring network time
- Lock deferral
- UNION ALL performance optimization
- Trusted stored procedure with embedded SQL
- Buffer pool memory allocation
- Improved concurrency through lock deferral
- Table append performance improvements
- Improved sort performance

Most of these enhancements do not require any user action to benefit from the performance gains, and we will only briefly describe those that need the user to take some action to achieve performance benefit:

A.4.1 Backup compression

DB2 UDB Version 8.1.4 now provides a backup compression technique that will allow database administrators to follow a robust recovery policy while minimizing the number of disks and tapes that must be dedicated to database backups.

While this is categorized as a manageability enhancement, the reduction in the size of the backup image can reduce I/Os and improve the performance of `backup` and `restore`.

A.4.2 Range-clustered tables

Range-clustered tables (RCT) provide fast, direct access to data.

An RCT is a table layout scheme where each record in the table has a predetermined offset from the logical start of the table. An algorithm is used to equate the value of the key for the record with the location of a specific row within a table. The basic algorithm is fairly simple. In its most basic form (using a single column instead of two or more columns to make up the key), the algorithm maps a sequence number to a logical row number. The algorithm also uses the record's key to determine the logical page number and slot number. This process provides exceptionally fast access to records; that is, to specific rows in the table. Using the key, the exact logical location of the record on disk can be found using linear interpolation. The algorithm is more complex when multiple columns are used to create the key.

The algorithm does not involve hashing because hashing does not preserve key-value ordering. Preserving key-value ordering is essential because it eliminates the need to reorganize the table data over time.

The advantages of an RCT include the following:

- Direct access through a range-clustered table key-to-RID mapping function.
- Less maintenance since a secondary structure such as a B+ tree does not need to be updated for every INSERT, UPDATE, or DELETE.
- Less logging is done for range-clustered tables when compared to a similarly sized regular table and associated B+ tree index.
- Less locking since an RCT avoids the redundancy of key and RID locking during searched updates, searched deletes, and inserts. Using an RCT in this way might reduce the number of locks and increase concurrency.
- Less buffer pool memory is required since there is no additional memory required to store a secondary structure.

A.4.3 Direct I/O support on AIX

Direct I/O is an alternative caching policy that bypasses the default file system buffer caching. Direct I/O reduces CPU utilization for file reads and writes by eliminating a data copy from the cache to the user buffer. It also avoids diluting the effectiveness of caching of other files in the case of a poor cache hit ratio.

DB2 UDB Version 8.1.4 provides direct I/O support on the AIX platform. This new support is for all SMS tablespace containers with the exception of Long Tablespaces and temporary tablespaces. The **`DB2_DIRECT_IO`** registry variable must be set to enable this function.

Note: On the Windows platform, direct I/O support is already available for all SMS and DMS containers with the **`DB2NTNOCACHE`** registry variable.

A.4.4 Asymmetric index splitting

In Version 8.1.4, the **`CREATE INDEX`** statement has been enhanced to provide greater control over the way index pages are split. The way in which the index pages are split can have a direct impact on the amount of space used by the index for any given table. The default 50/50 split is most effective for random table insertion patterns. The new clauses on the **CREATE INDEX** statement allow the index split method to be configured to match more defined insertion patterns via the **PAGE SPLIT SYMMETRIC** (default), **PAGE SPLIT HIGH** and **PAGE SPLIT LOW** clauses.

A.4.5 Buffer pool memory allocation

Starting in Version 8.1.4, the size for buffer pool memory allocations can be set using the **DB2_ALLOCATION_SIZE** registry variable. Setting this variable to a higher value results in fewer allocations needed to reach the desired amount of memory allocated to a buffer pool. The potential cost of setting a higher value for this registry variable is that memory can be wasted if the buffer pool is altered by a non-multiple of the allocation size. For example, if the value for **DB2_ALLOCATION_SIZE** is 8 MB and a buffer pool is reduced by 4 MB, this 4 MB will be wasted because an entire 8 MB segment cannot be freed.

A.4.6 Page cleaning enhancements

An alternative method of configuring the page cleaning system is supported, which differs from the default behavior in that page cleaners behave more proactively in choosing which dirty pages get written out at any given point in time. This new method of page cleaning differs from the default page cleaning in two major ways:

1. Page cleaners are not triggered by the **chngpgs_thresh** database manager configuration parameter.
2. Page cleaners are not triggered by LSN gap triggers issued by the logger.

To use this new method of cleaning, the **DB2_USE_ALTERNATE_PAGE_CLEANING** registry variable must be set to ON. When this variable is set to ON, DB2 uses a proactive method of page cleaning, writing changed pages to disk, keeping ahead of LSN gap, and proactively finding victims. Doing this allows the page cleaners to better utilize available disk I/O bandwidth.

A.4.7 Lock deferral

To improve concurrency, DB2 now permits the deferral of row locks of CS or RS isolation scans in some situations until a record is known to satisfy the predicates of a query. By default, when row-locking is performed during a table or index scan, DB2 locks each row that is scanned before determining whether the row qualifies for the query. To improve concurrency of scans, it may be possible to defer row locking until after it is determined that a row qualifies for a query.

To take advantage of this feature, enable the **DB2_EVALUNCOMMITTED** registry variable. With this variable enabled, predicate evaluation can occur on uncommitted data, which means that a row may not satisfy the query whereas if the evaluation occurred against the actual committed data, the row would satisfy the query. Additionally, uncommitted deleted rows are skipped during table scans. DB2 will skip deleted keys in type-2 index scans if the **DB2_SKIPDELETED** registry variable is enabled.

These registry variable settings apply at compile time for dynamic SQL and at bind time for static SQL. This means that even if the registry variable is enabled at runtime, the lock avoidance strategy is not employed unless **DB2_EVALUNCOMMITTED** was enabled at bind time. If the registry variable is enabled at bind time but not enabled at runtime, the lock avoidance strategy is still in effect. For static SQL, if a package is rebound, the registry variable setting at bind time is the setting that applies. An implicit rebind of static SQL will use the current setting of the **DB2_EVALUNCOMMITTED**.

A.4.8 Improved sort performance

Performance of sort operations, particularly small sorts and sorts running on Windows platforms, has been enhanced. These performance improvements are realized by minimizing the creation and deletion of temporary files used during sort operations. Temporary tables in SMS tablespaces will not be deleted by default once they are no longer needed. Instead, files associated with temporary tables that are larger than one extent in size will be truncated to one extent.

You can increase this amount by specifying a larger value for the **DB2_SMS_TRUNC_TABLE_THRESH** registry variable. You might want to increase this value if your workload repeatedly uses large SMS temporary tables and you can afford to leave space allocated between uses.

This feature will benefit users whose workload involves dealing with many small temporary tables on smaller systems such as Windows NT where the file system calls are relatively expensive, and users whose disk storage is distributed, requiring network messages to complete file system operations.

Note: You can turn off this feature by specifying a value of 0 (zero) for the **DB2_SMS_TRUNC_TABLE_THRESH** registry variable. You might want to do this if your system has restrictive space limitations and you are experiencing repeated out of disk errors for SMS temporary tablespaces.

If this variable is set to -1, then the file is not be truncated at all and the file will be allowed to grow indefinitely, restricted only by system resources.

The first connection to the database deletes any previously allocated files. If you want to clear out existing temporary tables, you should drop all database connections and reconnect, or deactivate the database and reactivate it.

If you want to ensure that space for temporary tables stays allocated, use the **ACTIVATE DATABASE** command to start the database. This will avoid the repeated cost of startup on the first connect to the database.

B

Workloads used in the scenarios

In this appendix we describe the workloads, scripts, and configurations used in the problem determination scenarios.

B.1 Introduction

The three main applications we used in our scenarios were the following:

- DTW workload
- EBIZ database
- Trade 2 application

We also used the WebSphere Performance Tool (WPT) which includes **`akstress`**, which we used to drive our workload.

These are briefly described here.

B.2 DTW workload

The DTW workload is a multi-user OLTP workload simulating an order entry system. It models a warehouse order entry system. The workload tool, called **`drvdtw`**, is built from C code and stored procedures. It generates OLTP transactions against the database from a specified number of clients. The transaction mix includes selects, inserts, and updates. The tool takes a number of command options that control its behavior.

B.3 EBIZ database

The EBIZ database is part of the EZMart application, which is a data warehousing solution for small-to-medium banks. The EZMart application is based on the EBIZ database. The database stores detailed information about customers: their accounts and transactions, as well as branch, product, and time information.

For the purposes of the performance scenarios in this redbook, we used a subset of the tables for various BI like queries.

The CUSTOMER table contained 600000 rows.

B.4 Trade2 database and application

The Trade 2 benchmark, also called the WebSphere Performance Benchmark Sample, has been developed by IBM and is publicly available. This application models an online brokerage firm providing Web-based services such as login, buy, sell, get quote, and more. Figure 6-65 shows the various application components and model-view-controller topology.

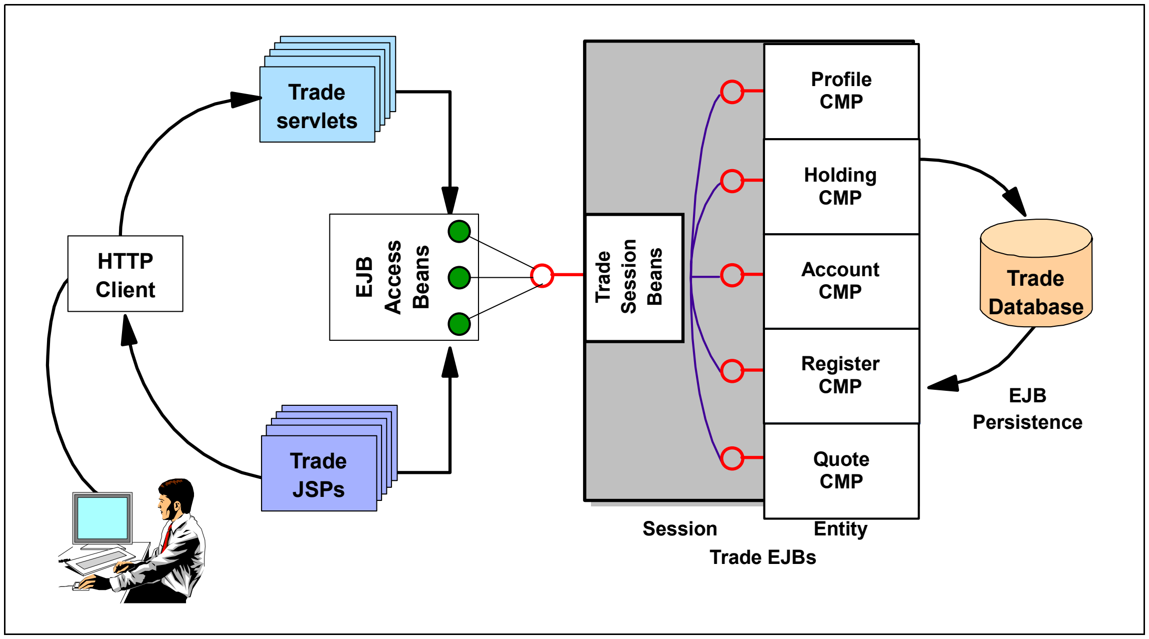


Figure 6-65 Trade 2 application

The Trade 2 application is a collection of Java™ classes, Java Servlets, Java Server Pages and Enterprise Java Beans (EJBs) that service requests made by registered users. This application runs as a single Java process that is managed by WebSphere Application Server.

This workload exercises the entire solution stack, which consists of the WebSphere Application Server, JVM, and the Just-In-Time (JIT) compiler, the HTTP server, the DB2 Database Server and the DB2 client, the AIX operating system, and the system hardware. The database contains approximately 500 registered users and 500 stocks.

Further details about this application, including sample code, can be obtained from:

`http://www.ibm.com/software/webservers/appserv/wpbs_download.html`

B.5 WebSphere Performance Tools (WPT)

WPT (formerly AKtools) is a set of applications allowing a user to test a Web server, a Web site, and/or a Web application.

Attention: This product is currently available as an IBM internal use only tool.

Version 1.9 of WPT consists of two applications:

- **`akstress`** is a high-performance, simple, threaded HTTP engine which is capable of simulating hundreds or even thousands of HTTP clients, using a highly configurable set of directives in a human readable and easily modified configuration file.
- **`akrecord`** is a simple eavesdropping proxy that will record a user's session against a Web server for later playback in **`akstress`**.

When the two applications are combined, it becomes very easy to quickly build an **`akstress`** configuration, which, with minor tuning, allows a user to evaluate the usability of a server, site, or Web application.

`akstress` is built on the code from several other internal IBM stress test tools. Those tools have been used for the last several years for things like HTTP/1.1 verification testing, large Web site stress analysis, HTTP Server SVT testing, and Web server unit testing efforts.

The following is a list of some of the available functions in this tool:

- Fully configurable HTTP headers
- SSL support
- Support for HTTP/1.1 functions, including persistent connections and chunked-transfer encoding
- Built-in cookie cache (for session testing)
- Result verification
- Full logging
- Overall and request-level statistics
- Simple to use, no requirement for third party interpreters, and so on
- Socks support for recording and replay

Attention: It is important to note that WPT is *not* a replacement for some of the high-end Web stress tools. It was created to be either a “quick and dirty” testing tool, or used in environments where the cost of purchasing of high-end tools is prohibitive.

Further details can be obtained from:

```
http://www.alphaworks.ibm.com/tech/wptools
```

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

IBM Redbooks

For information on ordering these publications, see “How to get IBM Redbooks” on page 493. Note that some of the documents referenced here may be available in softcopy only.

- *DB2 UDB Performance Expert for Multiplatforms - A Usage Guide*, SG24-6436
- *Database Performance Tuning on AIX*, SG24-5511
- *DB2 UDB V7.1 Performance Tuning Guide*, SG24-6012
- *The POWER4 Processor Introduction and Tuning Guide*, SG24-7041
- *DB2 UDB/WebSphere Performance Tuning Guide*, SG24-6417
- *IBM ESS and IBM DB2 UDB Working Together*, SG24-6262
- *DB2 UDB’s High Function Business Intelligence in e-business*, SG24-6546
- *DB2 UDB Exploitation of the Windows Environment*, SG24-6893
- *Scaling DB2 UDB on Windows Server 2003*, SG24-7019
- *DB2 Evaluation Guide for Linux and Windows*, TIPS-0142

Other publications

These publications are also relevant as further information sources:

- *DB2 UDB What's New Version 8*, SC09-4848
- *DB2 UDB Administration Guide: Planning*, SC09-4822
- *DB2 UDB Administration Guide: Implementation*, SC09-4820
- *DB2 UDB Administration Guide: Performance*, SC09-4821
- *DB2 UDB SQL Reference Volume 1*, SC09-4844
- *DB2 UDB SQL Reference Volume 2*, SC09-4845

- Date, C. J., *An Introduction to Database Systems*, Eighth Edition, Pearson Addison Wesley Publishing Co., July 2003, 0-321-197844-4
- IBM Education, “C45 DB2 UDB V8 Performance Tuning and Monitoring Workshop”
- IBM Education, “CF41 DB2 UDB Advanced Administration Workshop”
- “DB2 UDB Internals for Administrators”, M. Huras, IDUG Conference Proceedings, May 2003
- “DB2 V8 for Linux, Unix and Windows Performance and Tuning”, Christopher Tsounis, IDUG Conference Proceedings, May 2003
- “Materialized Query Tables”, W. O’Connell, IDUG Conference Proceedings, May 2003
- “New Data Clustering Techniques for Performance: Multidimensional Clustering”, L. Cranston, IDUG Conference Proceedings, May 2003
- Gunning, P. K., *DB2 UDB V8 Handbook for Windows and UNIX/Linux*, Prentice Hall PTR, August 2003, ISBN 0-130-66111-2
- “Boosting Query Performance: Multidimensional Clustering”, Bishwaranjan Bhattacharjee, Leslie Cranston, Tim Malkemus, Sriram Padmanabhan, Q2, 2003

  `http://www.db2mag.com/homepage.shtml`
- Gulutzan, P. and Pelzer, T., *SQL Performance Tuning*, AddisonWesley Publishing Co., September 2002, ISBN 0-201-79169-2
- “Everything You Wanted to Know About DB2 Universal Database Processes”, D. Snow and R. F. Chong

  `http://www7b.software.ibm.com/dmdd/library/techarticle/0304chong/0304chong.html`
- “Database Technology Leaps Ahead”, P. K. Gunning, Q4, 2002

  `http://www.db2mag.com/homepage.shtml`
- “Tuning up for OLTP and Data Warehousing”, S. Hayes and P. K. Gunning, Q4, 2002

  `http://www.db2mag.com/homepage.shtml`
- “Advanced Performance Diagnostics in DB2 for Unix, Linux and Windows”, Steve Rees, IBM Data Management Technical Conference Proceedings, September 2002
- “Maximize DB2 UDB for Linux, Unix, and Windows Performance -- Eliminate the Problems”, Melanie Stopfer, IBM Data Management Technical Conference Proceedings, September 2002

- "Where Did My Memory Go? - Memory Usage in UDB", Ian Maione, IDUG Conference Proceedings, October 2001
- "DB2 Tuning Tips for OLTP Applications", Yongli An and Peter Shum, IBM Toronto Lab, IBM Canada, July 2001
  http://www-900.ibm.com/developerWorks/cn/dmdd/library/techarticles/0107anshum/0107anshum_eng.shtml

Online resources

These Web sites and URLs are also relevant as further information sources:

- The IBM DB2 Technical Support site with pointers to documentation, downloads and other support options.

  http://www-3.ibm.com/cgi-bin/db2www/data/db2/udb/winos2unix/support/v8pubs.d2w/en_main
- The IBM DB2 Developer Domain site contains spec sheets, white papers, and technical and product documentation. The site includes a search engine.

  http://www7b.software.ibm.com/dmdd/library/
- International DB2 Users Group.

  http://www.idug.org/html/home.asp

How to get IBM Redbooks

You can search for, view, or download Redbooks, Redpapers, Hints and Tips, draft publications and Additional materials, as well as order hardcopy Redbooks or CD-ROMs, at this Web site:

http://www.redbooks.ibm.com/

Help from IBM

IBM Support and downloads. This site is a general entry point to all IBM support, with links to product documentation, downlaods, fixes, etc.

http://www.ibm.com/support/us/

IBM Global Services

ibm.com/services

Index

IBM

Redbooks

DB2 UDB ESE V8 non-DPF Performance Guide for High Performance OLTP and BI

(1.0” spine)
0.875”<->1.498”
460 <-> 788 pages

DB2 UDB ESE V8 non-DPF Performance Guide for High Performance OLTP and BI

Overview of DB2 UDB ESE V8 non-DPF architecture

Best practices for optimal performance

Problem determination scenarios

This IBM Redbook provides detailed information on implementing high performance OLTP and BI applications in DB2 UDB ESE V8 non-DPF environments involving AIX and Windows 2000 platforms. It is aimed at a target audience of experienced DB2 application developers and database administrators (DBAs).

We provide an overview of the architecture of a DB2 UDB V8 non-DPF environment from a performance perspective, and describe key performance drivers in OLTP, BI, and mixed workload environments.

This redbook's primary focus is a single partition (non-DPF) environment, and we provide best practices to achieve optimal application and system performance in OLTP, BI, and mixed workload environments.

Finally, we discuss some of the commonly encountered problems faced by a DBA when managing a DB2 environment, and describe techniques for problem diagnosis using typical problem scenarios.

SG24-6432-00 ISBN 0738498807