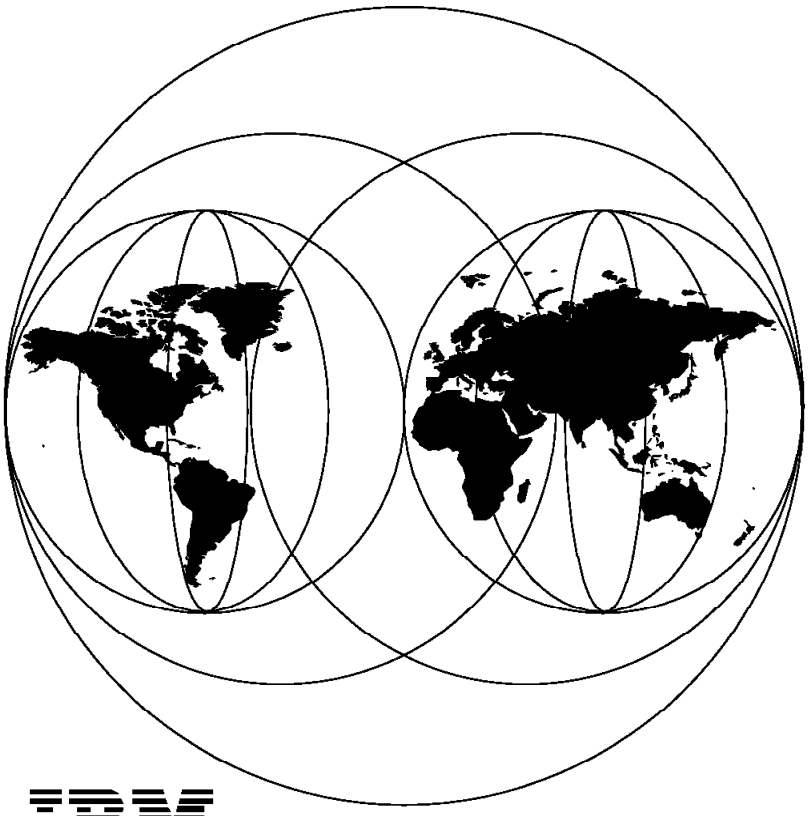
SG24-4814-00

DATABASE 2 Common Server Version 2.1.1 Performance and Tuning Guide

December 1996



IBM

International Technical Support Organization

Austin Center

International Technical Support Organization

SG24-4814-00

DATABASE 2 Common Server Version 2.1.1 Performance and Tuning Guide

December 1996

Take Note!

Before using this information and the product it supports, be sure to read the general information in Appendix B, “Special Notices” on page 133.

First Edition (December 1996)

This edition applies to DATABASE 2 Version 2.1.1 Common Server for use with the AIX, NT and OS/2 Operating Systems.

Comments may be addressed to:
IBM Corporation, International Technical Support Organization
Dept. JN9B Building 045 Internal Zip 2834
11400 Burnet Road
Austin, Texas 78758-3493

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

Contents

Figures

Tables

Preface

This redbook addresses configuration issues that may help you obtain the best performance from your database server.

This redbook was written for technical professionals who are involved in the administration or support of a DB2 Common Server installation.

There are a number of utilities that can be used in diagnosing the performance and database activity. This book looks at each of these and outlines the different methods that you can use to further tune your database environment.

Some knowledge of database environments, as well a basic understanding of the activity that is being performed in the database environment, is assumed.

How This Redbook Is Organized

This redbook is organized as follows:

- Chapter 1, “Performance and Tuning Overview” on page 1

  This provides and overview of the products, network configurations and systems that are being used throughout this book.

- Chapter 2, “Database Design” on page 5

  This chapter looks at the individual design of a database and what factors need to be considered to get the best performance possible.

- Chapter 3, “DB2 Performance Considerations” on page 11

  Tuning the database engine is a key way to obtain better performance. This chapter discusses the different tuning parameters available and the effects that they will have on the database environment.

- Chapter 4, “Database Monitoring” on page 45

  If you are suffering from a performance problem, the first step in resolving it is to find out exactly where it is occurring. By monitoring the different activity in the database you may be able to pin-point the problem. This chapter looks at the tools available to help you monitor database activity.

- Chapter 5, “DB2 Tools and Utilities” on page 79

  There are a number of different tools that can be used when tuning your database environment. This chapter discusses the different tools available and how they can be used.

- Chapter 6, “Tasks and Methodology” on page 113

  A sensible approach to diagnosing and correcting problems needs to be adopted. Otherwise, you may spend valuable time looking at areas that will have little impact on your situation. This chapter outlines different methods that can be used to help find problems and resolve them in the shortest possible time.

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 Austin Center.

Frank Rusconi, Project Leader
International Technical Support Organization, Austin Center

Miguel Robles, Author
IBM Spain

Meng Ping (Angela), Author
IBM China

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

Marcus Brewer, Editor
International Technical Support Organization, Austin Center

Rebeca Rodriguez, Editor
International Technical Support Organization, Austin Center

Bill Wilkins, DB2 Performance
IBM Toronto

Grant Hutchison, DB2 Client/Server Service
IBM Toronto

Comments Welcome

We want our redbooks to be as helpful as possible. Should you have any comments about this or other redbooks, please send us a note at the following address:

redbook@vnet.ibm.com

Your comments are important to us!

Chapter 1. Performance and Tuning Overview

This chapter provides an overview of the tasks and concepts involved in tuning your database environment to obtain optimal performance.

1.1 Overview

There are many different types of database environments in existence today. These environments vary from stand-alone systems to environments with many different database servers and clients running on multiple platforms. To perform performance analysis and tuning in these types of environments can be a difficult task and requires a good understanding of the environment as well as the tools that you will use.

This book will provide you with some guidelines about the tasks that are required for performance analysis and will discuss the methods that are available to do the analysis and tuning.

There are many factors that affect the performance of your database environment. These include:

- CPU

  Too many users or running applications on a CPU may cause overall system degradation.

- Memory

  Every user and application will use some of your system's physical memory. If you have insufficient memory in your system then you may find that application will fail, or your system will start to thrash.

- Disk

  I/O performance can play an important part in a database system. Too much activity on a single disk or I/O bus may cause performance degradation.

- Network

  In today's client/server environments, the network plays an important role. If the network is too slow then this may appear to the client as a performance problem at the server.

- Application

  An application that makes poor use of programming techniques, such as the use of compound SQL or stored procedures, may not be getting the best performance out of the database environment.

- Locking

  When the number of users increases, then the correct use of locking may affect concurrency and may also be reflected in the performance.

- Database Schema

  There are many methodologies that discuss the layout of data across databases and tables. In general, poorly designed databases may have a negative effect on the overall performance.

The first task you must perform when tuning your database environment is to determine if there is a performance problem and if so, where it is.

If you are dealing with a new database environment then you must decide how you are going to lay out your data across the system or systems. To determine this, you need to look at factors such as the amount of data you are going to be dealing with, the number of users accessing that data and the types of applications or queries that the users are going to be running.

1.2, “Sizing a Database Server” provides you with an overall idea on how you can start to determine the size of the database server, or servers, that you will require.

1.2 Sizing a Database Server

When you are planning a new database server, you need to take into account several factors. These include:

- The expected size of the database
- The maximum number of concurrent users
- The number of transactions per second or per day
- The types of transactions being executed
- Cost

Given the above factors you may decide that you need only one database server, or that you are going to need multiple servers. You may even find that you need a parallel server in your configuration.

Deciding on a database server or servers should also take into consideration the database design. The database design may be suited more towards a single parallel server, multiple serial servers or a combination of both serial and parallel servers.

1.2.1 Considerations

An important consideration when planning for your hardware is to allow for expansion. You should consider how large your database may grow in six to twelve months time, and the number of users that will be accessing your data at that time. If you don’t plan for this growth, then your system may reach physical limitations that might require hardware changes that will disrupt your daily operations.

CPU

There are many machines available that range in price and performance. For your database environment you should choose a processor that will give you enough computing power for your environment. This will depend on the type of processing that you will be performing.

For example, an environment which processes large amounts of statistical information stored in a database will not only require storage capacity, but may also require considerable processing capacity.

It is also possible that the processor you choose is guided by other factors such as I/O and storage scalability. When selecting the best processor for your

environment, you should check the price/performance information available at the time and make sure that you are allowing for scalability.

Disk

Purchasing enough disk for your environment is critical. Deciding on how much disk you are going to require will depend upon more than how large your database is.

The first use of disk is to contain the operating system. The size of the operating system will depend on which one you choose and the number of products that you install. The size of the operating system also depends on factors such as how much paging space you require and how you intend to partition the disks.

The next user of disk is the database products and instances. Appendix A, “DATABASE 2 Sizing Worksheets” on page 127 includes information about the disk usage for the different database components in DB2 for Common Server environments.

Finally, you should also consider how you are going to perform database transaction logging and backups of your data. Many environments require considerable disk space for transaction logs. The amount of disk space that you will require for your database logs is dependent upon both the size of the database and the types of transactions that are performed on it.

There is also the possibility that you will back up your database to disk, rather than to a device such as a tape drive. If this is the case, then you will probably want to make sure that you have enough capacity to perform a backup and that the location of the backup disks are not the same as the database disks.

Memory

Appendix A, “DATABASE 2 Sizing Worksheets” on page 127 also includes information about the amount of memory required by the different components in DB2 for Common Server.

There will be certain memory requirements by each user that logs in to the system and accesses the database. Each application connection will require memory, and there are also per-database requirements.

Most systems allow for a virtual memory space, or paging space, to be used. This means that if all the physical memory is in use, then some of it will be paged to disk while another application makes use of the physical memory. If the amount of physical memory is too small then you might find that the system spends more time paging information in and out of physical memory than it spends actually running the applications. This situation is called **thrashing**, and it can be avoided by increasing the amount of physical memory or decreasing the number of applications that are using up the system’s memory.

Database Environment

If you have multiple servers and clients, you will need to size each type of configuration that you have.

You may decide that in your database environment you are going to need a parallel server, or perhaps a cluster using High Availability Services (HACMP).

For sizing of these environments, you should refer to the planning guides supplied with the products you choose to use. Other useful references are listed in Appendix C, “Related Publications” on page 135.

Software

As mentioned in the hardware considerations, you are going to need to consider what applications and products you are going to use on your server.

There are multiple options that you may choose from. You may choose to use products or software for different types of platforms, for example; AIX, OS/2, NT, HP/UX or Solaris. There is also the choice of running the serial or parallel version of DB2. Finally you may choose to use some of the DB2 extensions, such as the Database Extenders.

All of these factors will have an impact on the types and configuration of the machines or platforms you choose.

Software requirements are shipped with the products when purchased. Appendix A, “DATABASE 2 Sizing Worksheets” on page 127 includes some information about the amount of disk required by the components in DB2 for Common Server environments.

Chapter 2. Database Design

This chapter discusses the design of database environments and how configuration of tablespaces, indexes and data types may also affect performance. We discuss the different options available and how you may configure your environment for the best overall performance.

2.1 Database Instances

A database **instance** is the environment that may contain multiple databases. When you create a new database, it will be created under your current instance. The current instance is defined by the environment variable, DB2INSTANCE. On UNIX platforms this will map to a defined User ID, which is then referred to as the Instance Owner.

DB2 for Common Server installs configuration information about each instance under the instance owner's home directory. Additional environment information will be found in the following files:

- For Korn Shell or Bourne Shell

  ~$DB2INSTANCE/sqllib/db2profile

- For C Shell

  ~$DB2INSTANCE/sqllib/db2cshrc

The notation, ~$DB2INSTANCE, indicates the instance owner's home directory. Each user that wishes to be able to access the databases contained within this instance must have the environment variables set in his or her profile.

DB2 for Common Server allows multiple database instances to exist on a single machine. Each DB2 instance will consist of the following:

- An Instance Owner (UNIX platforms)
- A System Administration Group
- A System Maintenance Group
- A System Control Group
- Databases
- Database Managaer Configuration File

There are a number of configuration parameters that can be configured at the instance level. These parameters may be configured using the Command Line Processor or the Database Director utility as shown in Figure 1 on page 6. Tuning the parameters for one instance will not directly affect another instance on the same machine. However, you may find that freeing up resources, such as memory, in one instance may make the memory available to other instances.

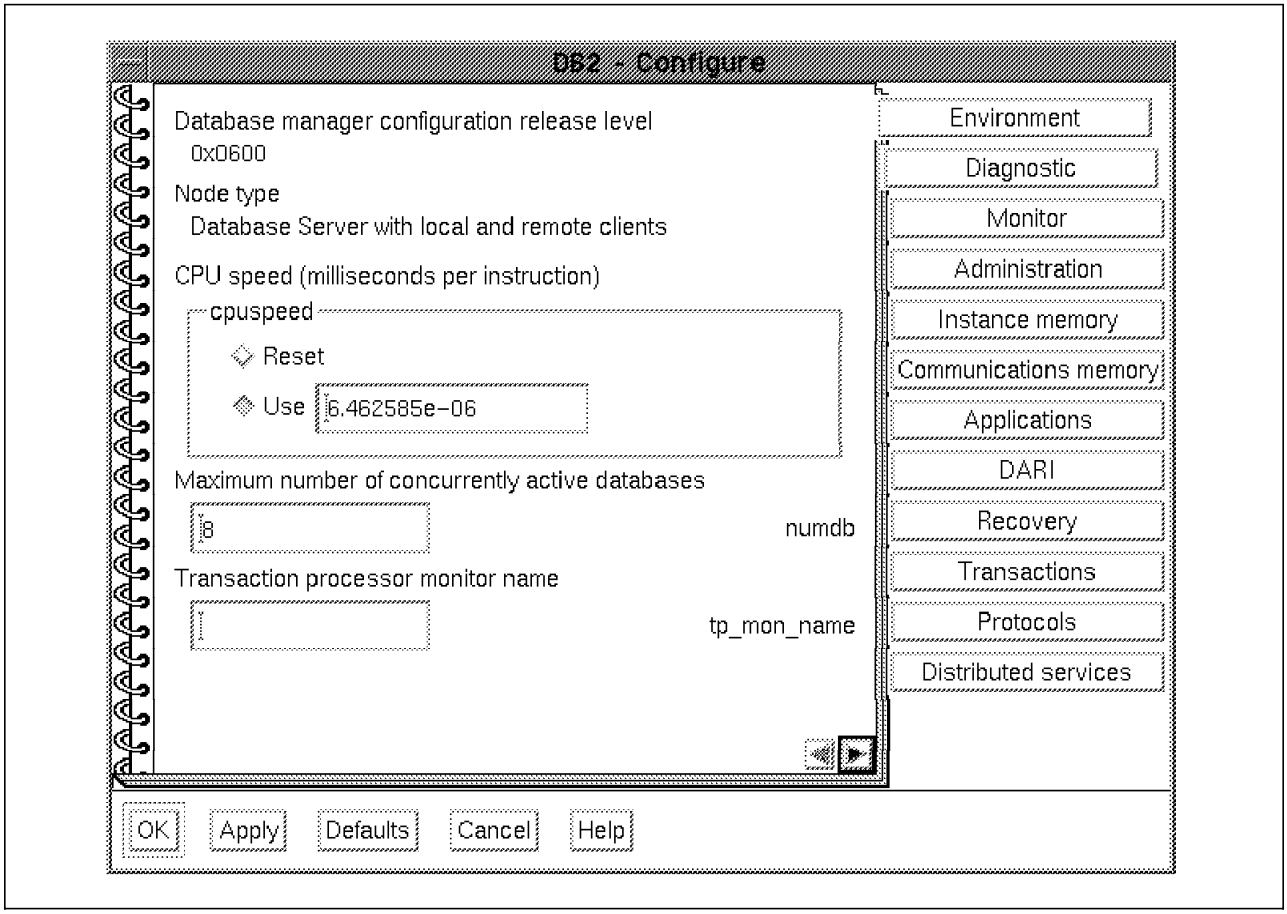


Figure 1. Database Director - Instance Configuration

The configuration parameters at the instance level affect all the databases contained within the instance.

Maintaining multiple instances has some overhead in terms of memory and the number of processes running on the machine. However, you need to consider that if you are connecting to a database in another instance, then your connection is treated as a remote connection and will have the associated overhead of network connections.

A possible reason for maintaining separate instances on a single machine could be administrative requirements. It is possible that you wish someone other than the instance owner to have system administration authority for a set of databases. However, this same person should have no special authority over a different set of databases. The only way to do this is to have multiple instances and add the user to the system administration group for the appropriate instance. While it is possible to have the same system administration group for both instances, in this example we would have a different administration group for the second instance.

Figure 2 on page 7 shows an example of multiple instances on a single platform. In this example, the first instance *(inst1)* could be a production system while the second instance *(inst2)* could be the development system. We can see from the figure that both instances contain different copies of the databases dbaseA and dbaseB. However, inst2 also contains an additional database, dbaseE.

We can also see that the example configuration parameters are set differently for the instances. The production instance has a different administration group from the development instance. Also, the authentication method used on the production instance is SERVER, which is more secure than the CLIENT authentication. Finally, the development instance has a higher diagnostic level set and has turned on statement monitoring, which can help in the development of new applications.

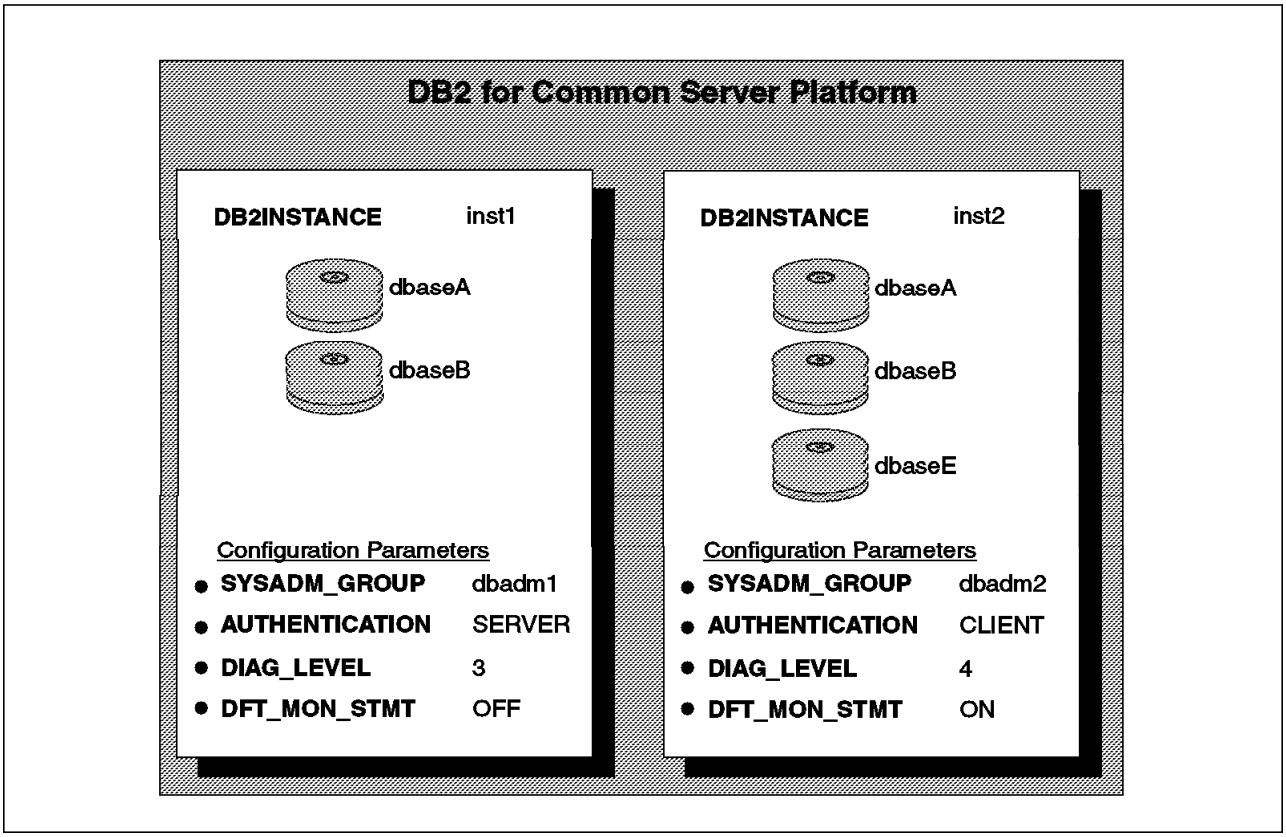


Figure 2. Multiple Instances

Information on tuning the instance level parameters is discussed in Chapter 3, "DB2 Performance Considerations" on page 11.

2.2 Tablespaces

Once you have decided on whether you wish to break the databases into a single or multiple instance, you will now need to determine how the tablespaces will be laid out. This may depend on the size and type of data that will exist in the database and will also depend on the disk or disks that are available.

One of the advantages in maintaining multiple tablespaces is that the data can be logically divided up and then backed up, and recovered, as required at the tablespace level. There is also the ability to use different types of tablespaces. Some tablespaces may contain large objects such as image or audio data, while other tablespaces may contain more traditional data such as tables containing character or integer columns. There may be some tables that are infrequently accessed and so may be placed on slower disks than data that is accessed more frequently.

Each database on your system will consist of at least three tablespaces. These are shown in Figure 3 on page 9 and include:

1. System Catalog Tablespace

   This system catalog tablespace is used for all the database system tables. The default name of the system catalog tablespace is SYSCATSPACE, and it is created when the database is created.

2. Temporary Tablespace

   Temporary tables are stored within the temporary tablespace. The default temporary tablespace created when the database is created is called TEMPSPACE1. You may add or delete temporary tablespaces, but at least one must exist at all times.

3. User Tablespace

   The user tablespace contains all the user tables and information. Again, a default user tablespace is created when the database is created and is called, USERSPACE1. This tablespace may be dropped and additional user tablespaces may be added to or deleted from the database.

In addition to these tablespaces, you are also able to create additional tablespaces. The three types of tablespaces that you can create are:

1. Regular Tablespace

   A regular tablespace will contain most of your database tables and indexes.

2. Long Tablespace

   If your database contains large objects stored as LONG or LOB data, then you may wish to store these columns in a special tablespace. This allows you to configure the tablespaces to allocate space and retrieve information in a more efficient manner for the types of data that they contain.

3. Temporary Tablespace

   Occasionally you may find that you require an additional temporary tablespace. You need to specify that a tablespace is a temporary tablespace when it is created.

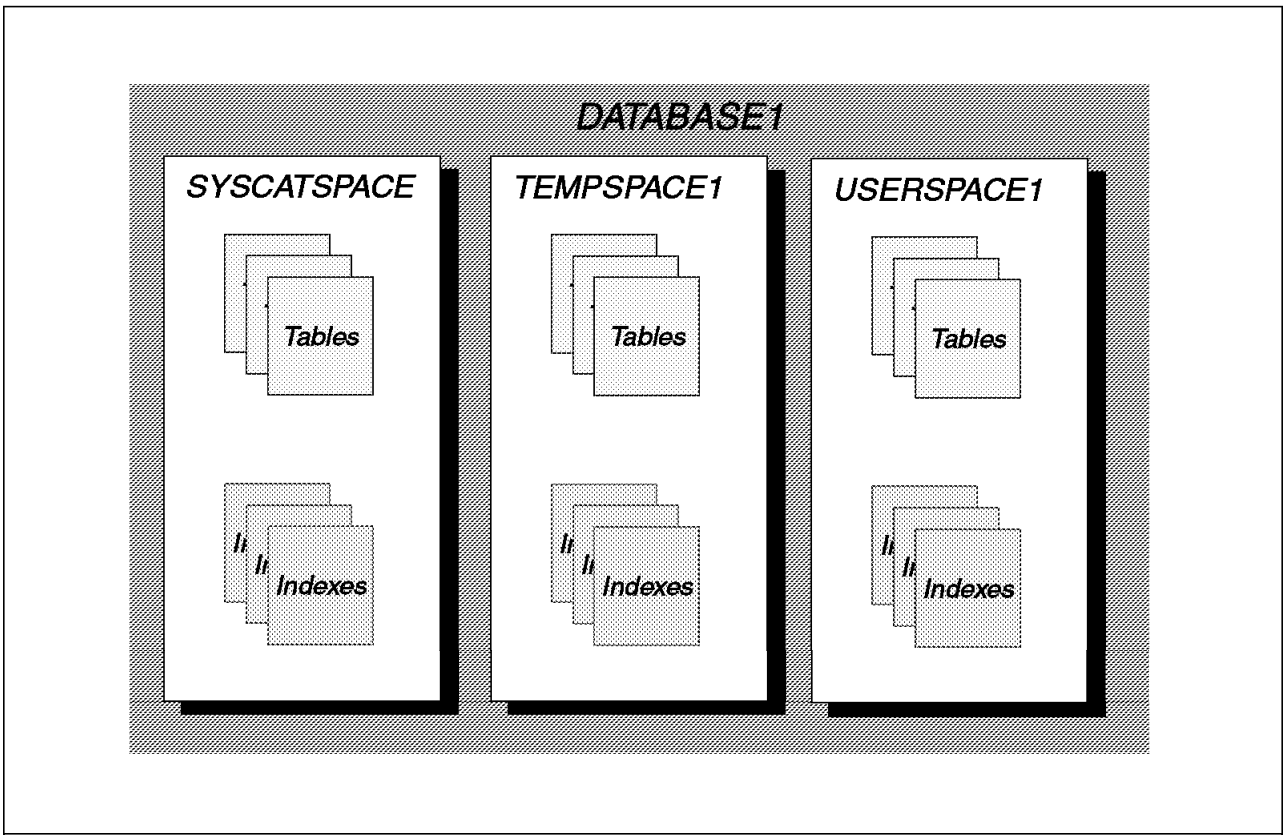


Figure 3. Database Tablespaces

DB2 for Common Server supports two different tablespace mechanisms. These are:

- System Managed Tablespaces
- Database Managed Tablespaces

You will have different considerations depending upon the type of tablespace you choose. It is possible to use a mixture of the different types of tablespaces.

Chapter 3. DB2 Performance Considerations

The overall performance of a database depends upon the following:

- The physical design of the database
- The logical design of the database
- The performance of applications that access the database
- The configuration of the database and DB2 parameters

Throughout this chapter, the database configuration parameters that have an impact on performance are reviewed. Parameters that affect system resources and database operation can be defined at either of the following:

1. Instance Level

   When an instance is created, a DBM configuration file is also created.

2. Database Level

   A database configuration file is created when each database is created.

These parameters can be changed from the system default values to obtain better performance or to increase the capacity of the database. Tools that can be used to measure the performance of your database environment are discussed throughout the following chapters.

Performance information is collected through 'data elements' that can be defined as gauges, counters, water marks, timestamps or information text. For a complete reference of these data elements, refer to *DB2 Database System Monitor Guide and Reference - for common servers* (S20H-4871).

Configuring database parameters can improve performance by:

- Reducing I/O access through buffering and caching
- Sizing the buffer pools, heaps and stacks used by the database
- Using logging more efficiently

There are several memory areas allocated for each database contained within the database manager instance. These memory areas include:

- Database Global Memory area

  This area is used for all the applications that might connect to the database.

- Agent Private Memory area

  This is used for each active or idle agent. A server agent is a process or thread that carries out all the requests made by a client application.

Figure 4 on page 12 shows the memory areas allocated in the server for a given database. The number of active agents will depend upon the number of applications connected to the database. Each connected application is served by its own agent process or thread. The number of idle agents will depend on configuration parameters of the database.

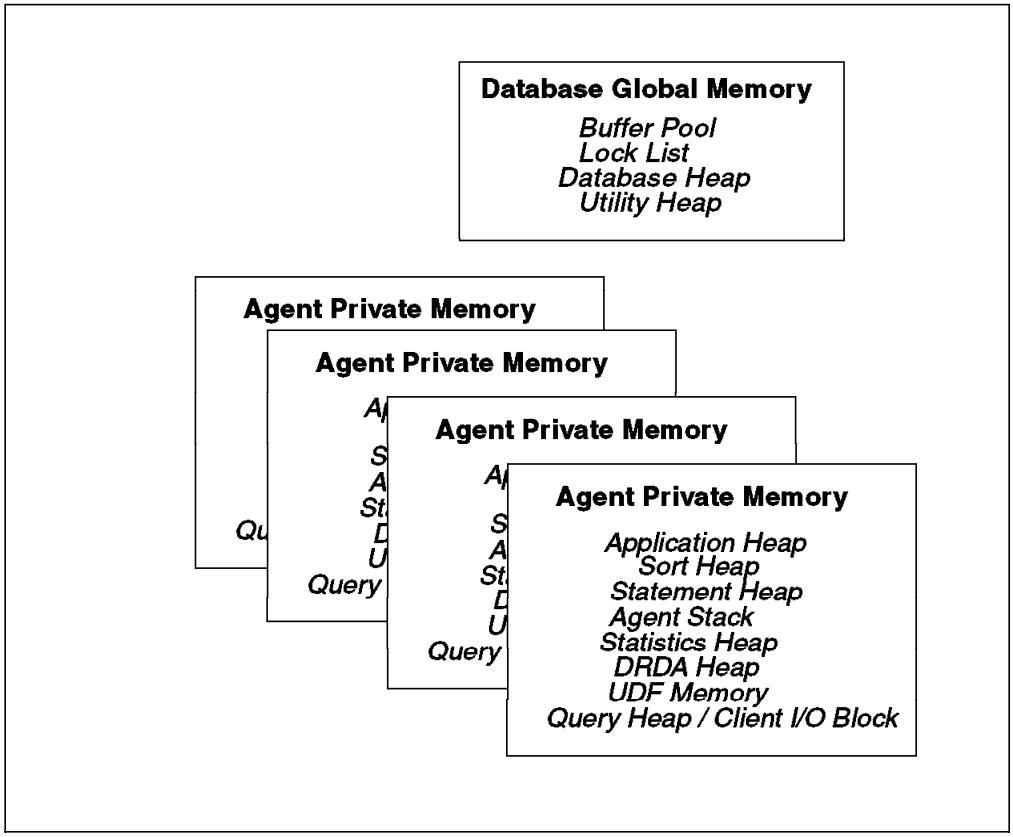


Figure 4. Memory Allocation on Server Platform

Most of the heaps, stacks and memory areas are only allocated when required. The only areas that remain allocated are the Buffer Pool and the Lock List.

Every instance may have several active databases. The maximum number of active databases for a given instance is limited by the numdb database manager configuration parameter. A single machine may also have multiple instances running simultaneously. You should note that resources allocated to one instance are not available to other instances. This is important, as allocating excessive resources, such as memory, to one instance may affect the performance of another instance on the same system.

3.1 Tuning Database Global Memory

Each database has its own Global Memory. This memory area contains the following elements:

- The Buffer Pool

  This memory area is used by the database for data caching. In this area, data pages are read and modified. Adjusting the size of this area can greatly effect performance.

- The Lock List

  This memory area is used by the database to store all the locks held by all the applications connected to the database.

- The Database Heap

The Database Heap contains control block information for tables, indexes and tablespaces. It includes the Catalog Cache and the Log Buffer.

- The Utility Heap

  The Utility Heap is used by the LOAD, RESTORE and BACKUP utilities.

The Buffer Pool and the Lock List areas are allocated when the first application connects to the database. The Database Heap and the Utility Heap grow as required, and only the maximum values for these heaps are configured. This is shown in Figure 5.

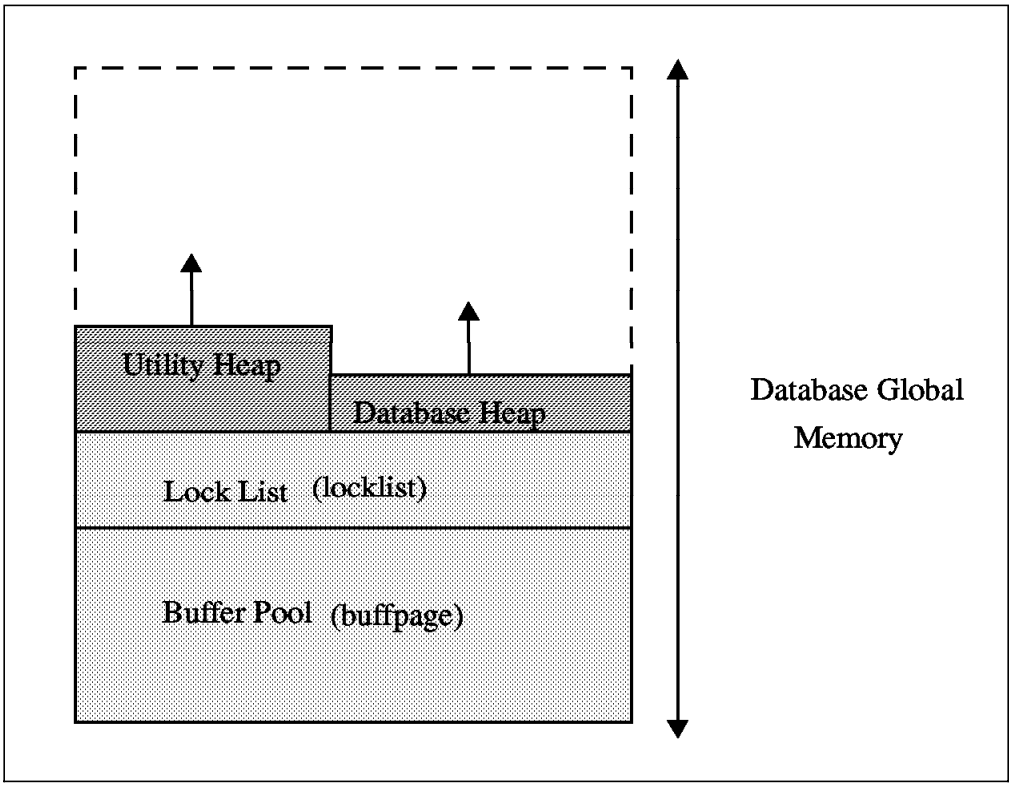


Figure 5. Database Global Memory

3.1.1 Database Buffer Pool

The database server reads and updates all data from the buffer pool. Avoiding disk I/O is the main issue when trying to increase the performance of the server. The size of the buffer pool is determined by the buffpage parameter. Proper configuration of the buffer pool has a large impact on the database performance.

This buffer pool is allocated when the database is activated. Data is copied from disk to the buffer pool as is required by applications. The “life cycle” of a page in the buffer pool is dictated by the following processes:

- Pages are placed in the buffer pool by I/O servers
- Pages are removed from the buffer pool by page cleaners (I/O cleaners)

If the server needs to read a page of data, and that page is already in the buffer pool, it will be able to access that page much faster than if the page was out on disk. It is then desirable to ‘hit’ as many pages as possible in the buffer pool. As data is frequently accessed through indexes, having a good index pool_hit

ratio and a good overall buffer pool_hit ratio will increase the performance of the database.

The following data elements can be measured by the DB2 performance tools to evaluate how the buffer pool is being used:

- Buffer Pool Data Logical Reads

  Total number of read data requests that went through the buffer pool.

- Buffer Pool Data Physical Reads

  Number of read requests performed that required I/O to place data pages in the buffer pool.

- Buffer Pool Index Logical Reads

  Total number of read requests for index pages that went through the buffer pool.

- Buffer Pool Index Physical Reads

  Number of read requests for index pages that require I/O activity to place index pages in the buffer pool.

An ‘index pool_hit’ ratio is calculated as the difference between the number of the index logical reads and the number of index physical reads divided by the total number of index reads requests. The following formula can be used to calculate this ratio:

$$\text{IndexPoolHitRatio} = \frac{\text{IndexLogicalReads} - \text{IndexPhysicalReads}}{\text{IndexLogicalReads}} \times 100$$

An overall ‘buffer pool hit’ ratio is calculated as the difference between the number of the all (data + index) logical reads and the number of all (data + index) physical reads divided by the total number of read requests.

$$\text{BufferPoolHitRatio} = \frac{\sum \text{LogicalReads} - \sum \text{PhysicalReads}}{\sum \text{LogicalReads}} \times 100$$

Care should be taken when evaluating these ratios. The number of Physical Reads includes the reads performed by prefetchers, so a low ‘buffer pool hit’ ratio may not indicate a performance problem. Prefetchers performing asynchronous I/O must be considered. Prefetchers, and their impact in the ‘synchronous buffer pool’ hit ratio are discussed in 3.1.1.2, “Prefetchers” on page 18.

The Index Pool Hit Ratio and the Buffer Pool Hit ratio are also influenced by data reorganization. Data reorganization, and its influence on the buffer pool, is discussed in 3.1.1.1, “Reorganizing Tables” on page 15.

The default size of the buffer pool is 1000 4KB pages for AIX servers and 250 4KB pages for OS/2 and NT servers.

3.1.1.1 Reorganizing Tables

To keep the database tuned, the reorg utility is one of the tools most frequently used. The reorg tool tries to store data rows in a clustering order. Rows are ordered according to an index, the clustering index. Only one index can be the clustering index. Reorganization of data or indexes may be necessary when:

- A table has been initially loaded with the load utility. Data loaded through the load utility is stored in the same order of the input file.
- Frequent updates to tables may cause the table's rows to be disordered very quickly. New rows are not inserted in a physical sequence with old rows, so more read operations, especially those involving a range of values, may be necessary to retrieve data.
- An index is defined, but the access plan shows that the database manager is not using it to access the data.

To execute the reorg utility you must have SYSADM, SYSMAINT, SYSCRTL or DBADM authority, or CONTROL privilege on the table. The reorg utility uses temporary tables while it reorganize the original table. Notice that reorg cannot be used on views. The database administrator needs to supply the following parameters to the reorg utility:

- Table name

  Fully qualified table name of the table to be reorganized.
- Index name

  Fully qualified index name to use when reorganizing the table.
- Tablespace name *(optional)*

  Name of the tablespace where the database manager will store the temporary table.

To decide when to reorganize a table or a set of tables, DB2 Common Server provides the reorgchk utility. To use the reorgchk utility you need SYSADM or DBADM authority, or CONTROL privilege on the table. reorgchk will base its results on statistics to determine if a reorganization is required. Those statistics can be the current statistics stored in the database system tables, or it can execute runstats to update table statistics and then use those updated statistics to decide on the need for a reorganization. The following is a sample of the output from the reorgchk utility. Notice that the output is divided into table and index statistics.

```
$db2 reorgchk update statistics on table db2a.customer

Doing RUNSTATS....

Table statistics:

F1: 100*OVERFLOW/CARD < 5
F2: 100*TSIZE / ((FPAGES-1) * 4020) > 70
F3: 100*NPAGES/FPAGES > 80

CREATOR  NAME                  CARD    OV    NP    FP    TSIZE  F1  F2 F3 REORG
-------------------------------------------------------------------------------
DB2A     CUSTOMER             75000     0  3453  3453 13650000   0  98 100 ---
-------------------------------------------------------------------------------
```

The output of the reorgchk utility shows the following statistics for the table:

- CARD

  The number of rows in the table.
- OV

  The number of overflowed rows. An overflowed row is an updated row that contains more bytes than the old row and does not fit in the original page. This happens when a column is added to a table or there is a VARCHAR field (and the updated row contains a longer value). In these cases, a pointer is kept at the row original location, so more I/O operations are required to retrieve the row.
- NP

  The number of pages that contain data.
- FP

  The total number of pages. Notice that a page can be empty after rows are deleted. Empty pages are read for a table scan. In our example, NP=FP so there are no empty pages.
- TSIZE

  Table size in bytes.
- F1, F2, F3

  Results of the three formulas. The meaning of each formula is as follows:
  - F1: The percentage of overflowed rows should be less than 5 percent.
  - F2: The size of the table should be more than 70 percent of the space allocated for the table.
  - F3: The percentage of empty pages should be less than 20 percent. reorg will reclaim empty pages and improve performance avoiding empty page reads during a table scan.
- REORG

  Three symbols, hyphens or asterisks, that indicate if the result of each formula is within the limits or not. A hyphen means that the result is within the limit and an asterisk indicates that it is not. The results for the previous example indicate that all formulae are within the limits. If the limits are exceeded then executing reorg utility will compact the table, recovering unused space and will reduce the number of I/Os required to retrieve data.

The second part of the output of the reorgchk utility is the index statistics. A sample of this output is shown below:

```
Index statistics:

F4: CLUSTERRATIO or normalized CLUSTERFACTOR > 80
F5: 100*(KEYS*(ISIZE+10)+(CARD-KEYS)*4) / (NLEAF*4096) > 50
F6: 90*(4000/(ISIZE+10)**(NLEVELS-2))*4096/(KEYS*(ISIZE+10)+(CARD-KEYS)*4)<100

CREATOR  NAME                 CARD  LEAF  LVLS ISIZE   KEYS   F4   F5  F6 REORG
------------------------------------------------------------------------------
Table: DB2A.CUSTOMER
DB2A     IDXACCT             75000   327     3     8  72441    4   98   62 *--
------------------------------------------------------------------------------
```

The statistics shown for every index are:

- CARD

  The number of rows in the table.
- LEAF

  The number of index leaf pages. This predicts the number of index page I/Os needed for an index scan.
- LVLS

  The number of index levels. This predicts the number of I/O operations required to access a specific index entry.
- ISIZE

  Index size.
- KEYS

  The number of unique index entries.
- F4, F5, F6

  Results of the three formulas for indexes. The meaning of these three formulas are:
  - F4: The clustering ratio of an index should be greater than 80 percent. In the previous example, it is only 4 percent. Notice that if there is more than one index for the table, only one of the indexes may have a good clustering ratio.
  - F5: Space allocated for indexes should be more than 50 percent occupied.
  - F6: The number of index entries should be more than 90 percent of the number of entries that NLEVELS-1 can handle.
- REORG

  The three symbols, hyphens or asterisks, that indicate if the result of each formula is within the limits or not. A hyphen means that the result is within the limit. An asterisk means that it is not. In our previous example, F4 is out of limits (F4=4 when the formula expects it to be greater than 80).

After executing reorg, reorgchk shows the following values for the index statistics:

```
Index statistics:

F4: CLUSTERRATIO or normalized CLUSTERFACTOR > 80
F5: 100*(KEYS*(ISIZE+10)+(CARD-KEYS)*4) / (NLEAF*4096) > 50
F6: 90*(4000/(ISIZE+10)**(NLEVELS-2))*4096/ (KEYS*(ISIZE+10)+(CARD-KEYS)*4)<100

CREATOR  NAME                 CARD  LEAF  LVLS ISIZE   KEYS   F4   F5  F6 REORG
------------------------------------------------------------------------------
Table: DB2A.CUSTOMER
DB2A     IDXACCT             75000   327     3     8  72441  100   98   62 ---
------------------------------------------------------------------------------
```

The reorg utility rearranges the table rows, so the statistics stored in the system tables will no longer show the real data distribution. Because of this, after using the reorg utility, runstats should be executed and applications affected should be rebound using the bind or rebind commands.

Table 1 on page 18 shows the difference for a query executed before and after reorganization. The query selected a set of rows through a predicate. The predicate involved a clustered index. Values for the data elements and response

time were obtained using the db2batch utility. The db2batch utility is discussed in 5.2, “Benchmarking Tool” on page 104. Values for Cluster Ratio were obtained using the utility, reorgchk.

	Before Reorg	After Reorg
Index Pool Hit Ratio	98.41%	45.45%
Buffer Pool Hit Ratio	4.01%	93.38%
Data Logical Reads	369	382
Data Physical Reads	354	20
Index Logical Reads	377	11
Index Physical Reads	6	6
Cluster Ratio	4%	100%
Response Time (seconds)	5.63	0.62

Table 1. Reorgs and Buffer Pool

Notice that the response time was reduced to a mere 12 percent of its value before reorganizing the table. The Buffer Pool Hit Ratio increased from 4.01 percent to 93.38 percent. The number of buffer pool Data Logical Reads is similar in both cases, but the number of costly buffer pool Physical Data Reads was greatly decreased.

Notice that after reorganizing the table, the database manager was able to retrieve the answer set with only twenty Data Physical Reads when before reorganizing, over three hundred Data Physical Reads were required. This clearly shows the benefits of clustering data.

The Index Pool Hit Ratio shown in Table 1 is misleading. Notice that number of Index Physical Reads in both cases was the same, 6. But after reorg, only 11 Index Logical Reads were necessary, while before reorg, 377 were performed. That’s why a higher Index Pool Hit Ratio is achieved before reorganizing the table (6 physical reads of 377 logical reads versus 6 physical reads of 11 logical reads).

3.1.1.2 Prefetchers

Prefetchers are I/O server processes (processes in AIX, threads in OS/2 and NT) that ‘read-ahead’ data for the database manager. Their goal is to place pages in the buffer pool before these pages are needed by the server. This is done in an asynchronous mode, as opposed to the synchronous I/O performed when a page is requested by an application. Asynchronous I/O can be overlapped with regular CPU activity.

Prefetching is activated by the database manager, depending on the SQL statement being executed. Sequential prefetching can be turned off using the seqdetect configuration parameter. The following data elements provide guidance about the effectiveness of prefetchers:

- Buffer Pool Asynchronous Data Reads

  Number of pages read asynchronously into the buffer pool.

- Buffer Pool Asynchronous Read Requests

  Number of asynchronous read requests.

If prefetchers are working properly, the percentage of synchronous read requests hitting pages in the buffer pool should be high. A synchronous buffer pool_hit ratio is calculated as the difference between the total number (index + data) of logical reads and the total number (index + data - async_data_reads) of synchronous physical reads divided by the total number of read requests.

$$\text{SyncBuffPoolHitRatio} = \frac{\sum \text{LogicalReads} - (\sum \text{PhysicalReads} - \text{AsyncDataReadReq})}{\sum \text{LogicalReads}} \times 100$$

Figure 6 plots the response time for a given query versus the size of the buffer pool. The response time improves when the size of the buffer pool is increased. However, a point is eventually reached where increasing the size of the buffer pool does not bring any performance benefit. Notice that the values shown here are for illustrative purposes only. It is the task of the database administrator to determine the point where an increase of the size of the buffer pool will not improve the performance of the database.

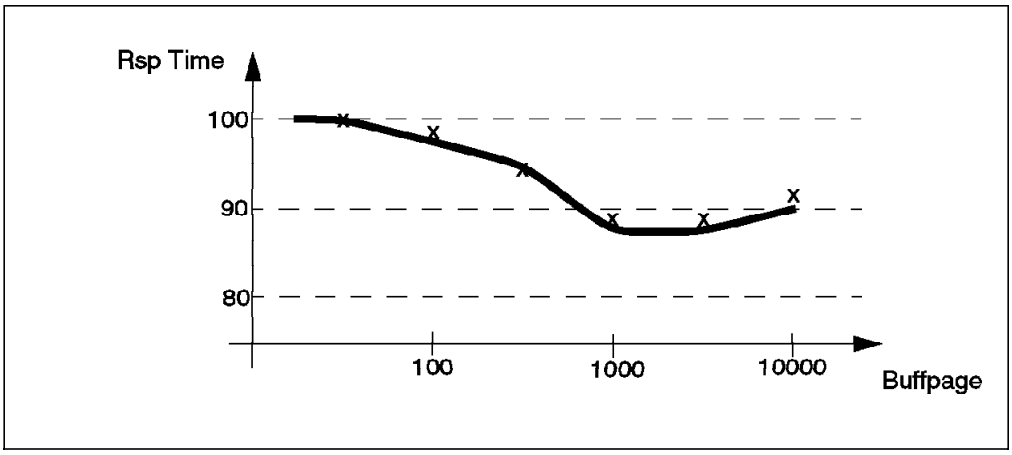


Figure 6. Buffer Pool and Response Time

A more significant conclusion can be reached if the response time is plotted versus the synchronous buffer pool hit ratio as shown in Figure 7 on page 20. When the synchronous buffer pool hit ratio gets close to 100 percent, the response time will not improve when more pages are added to the buffer pool. This is because, as you get close to the 100 percent ratio, all the pages required by the database have been previously placed in the buffer pool by the prefetchers. Almost all the read requests are performed asynchronously, and each read brings a number of pages into the buffer pool. The number of pages read in is determined by the prefetch size set for tablespace.

Notice that in the left side of the graph plotted in Figure 7 on page 20, the ratio decreases to 90 percent. This is caused by the fact that buffer pool did not have as many free pages available as the number of pages set by the prefetch size. No prefetching was then performed (no async reads at all). Even without prefetching, a 90+ synchronous buffer pool hit ratio is achieved. This 90+ synchronous buffer pool ratio comes from the fact that a physical read operation will place a 4KB page in the buffer pool, thus placing many rows in the buffer pool. Subsequent read requests will find rows already in the buffer pool, without requiring an additional I/O operation.

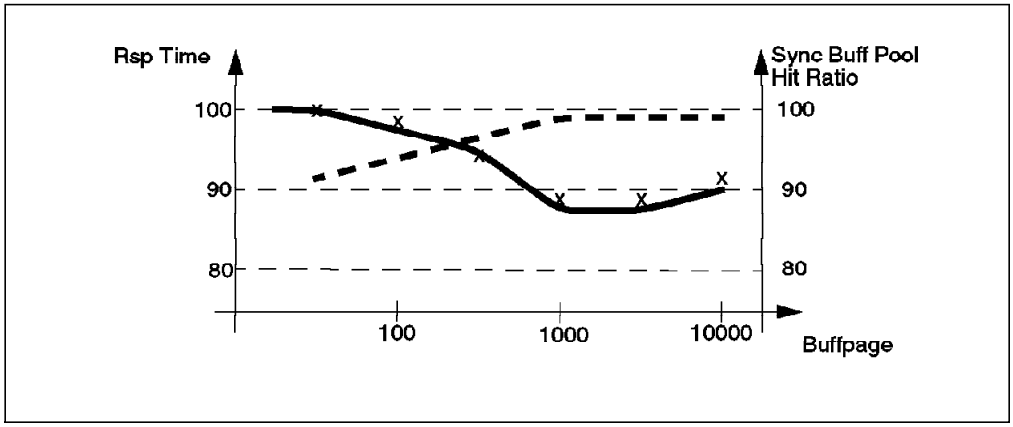


Figure 7. Response Time and Synchronous Buffer Pool Hit Ratio

Care should be taken when trying to increase the synchronous buffer pool hit ratio. If a big Prefetch Size is configured for tablespaces, this ratio will improve. But there is a risk that the prefetchers may provoke unnecessary I/O by reading too many pages, and thus, creating disk contention. Disk contention should be monitored using standard operating system tools, such as `iostat` for AIX servers. Steady values of synchronous I/O read time indicate no disk contention generated by prefetchers. The following data elements help determine the synchronous I/O read time:

- Buffer Pool Asynchronous Read Time

  Elapsed time spent reading by prefetchers.

- Total Buffer Pool Physical Read Time

  Elapsed time spent processing read requests that caused data or index pages to be physically read from disk to the buffer pool.

The difference between these two values divided by the number of synchronous read requests gives an estimate of the synchronous I/O read time.

$$\text{Sync I/O ReadTime} = \frac{\text{TotalBufPoolPhysReadTime} - \text{BufPoolAsyncReadTime}}{\sum \text{LogicalReads}}$$

3.1.1.3 Prefetchers and I/O servers

The number of processes, or threads, used to perform prefetching and other asynchronous I/O tasks, is limited by the `num_ioservers` database configuration parameter. By default, the number of I/O servers is set to three. The number of prefetchers should be related to the number of I/O devices, namely disks, where containers used by the database are stored.

Figure 8 on page 21 shows the response time of a query using different numbers of I/O servers. The query selected data from a table which has a tablespace spread across four containers. The table was large enough to have extents in all four containers. Each container was placed on a different disk drive and the table was the only table stored in the tablespace. A 2 percent gain was obtained when the number of I/O servers was increased from one to two.

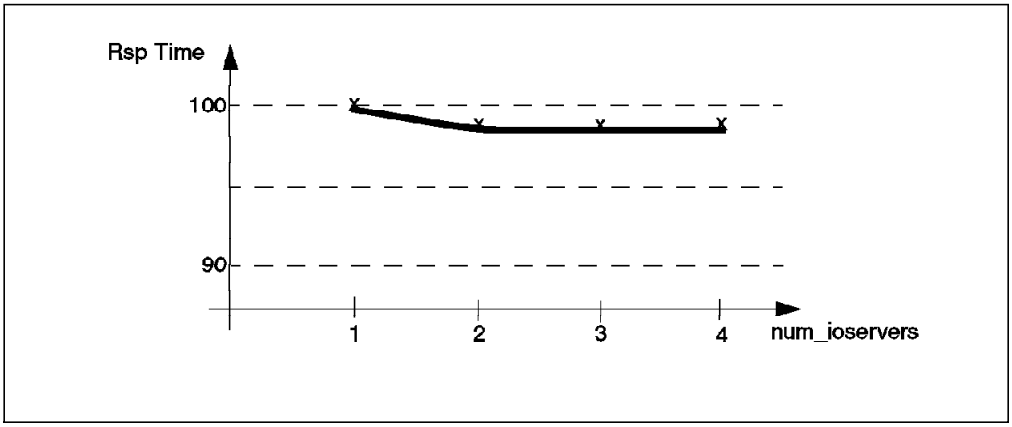


Figure 8. Response Time and Number of I/O Servers

3.1.1.4 Prefetch Size and Physical Reads

The amount of data that is 'read-ahead' depends on the Prefetch Size determined when the tablespace was created. The prefetch size for a tablespace can also be modified later using the `alter tablespace` command.

The value of the Prefetch Size states the maximum amount of pages that can be prefetched when retrieving data from the tablespace. Initially, the database manager will start with only a portion of the prefetch size. It will then continue to increase it until the buffer pool limit is hit (see `maxchngpgs`, discussed in 3.1.1.5, "Page Cleaners" on page 22). Once the limit is hit, the database manager will start to decrease the number of pages being prefetched.

Figure 9 shows the value of data elements captured through a snapshot. The snapshot monitor is discussed in chapter 4.2, "Snapshot Monitor" on page 47. The application monitored was a dynamic SQL statement retrieving 75000 rows.

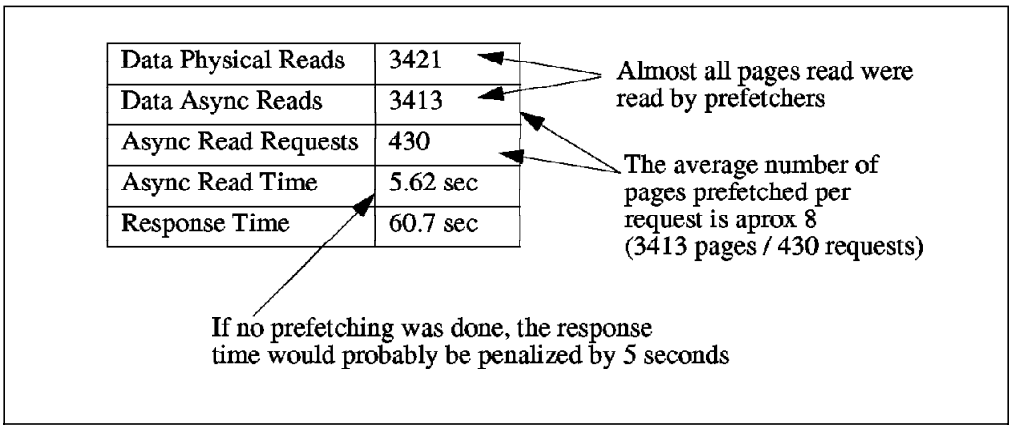


Figure 9. Efficiency of Prefetchers

Notice the number of data pages read asynchronously. All asynchronous reads are performed by prefetchers. Almost all pages are read asynchronously, achieving a Buffer Pool Hit ratio close to 100%. Prefetchers are placing nearly all the required pages in the buffer pool. The database engine will find them there and will not need to perform additional I/O operations to retrieve data.

In the example, each prefetcher reads, on average, eight pages into the buffer pool (even though the Prefetch Size was set to 32 pages). Since the prefetchers

read asynchronously, the time spent in these 430 async reads is time that we saved. If no prefetchers were used, the response time would probably go well over 65 seconds, as every read operation would only read one page at a time (and would not be asynchronous).

3.1.1.5 Page Cleaners

Page cleaners examine the buffer pool, and write pages asynchronously to disk. The page cleaners have two goals:

- Assure that an agent will always find free pages in the buffer pool. If an agent does not find free pages in the buffer pool, it will have to clean them itself, and the application being served by this agent will have a poorer response.
- Speed the recovery of a database if a system crash occurs. If more pages are written to disk, the amount of log file to be processed to recover the database will diminish.

Page cleaners are activated whenever any of these two events occur:

1. The number of written pages in the buffer pool exceeds the `maxchngpgs` database configuration parameter.

   `maxchngpgs` specifies the percentage of changed pages at which the asynchronous page cleaners will be activated. This database configuration parameter is, by default, set to 60 percent. Only modified pages are written to disk. Non-modified pages are replaced by new ones without any page cleaner intervention.

2. The size of the log that would need to be read to recover the database is greater than `logprimary * logfilsiz * softmax`.

   `logprimary` is the number of primary log files, `logfilsiz` is the size of the log files and `softmax` is the percentage of the log file used before taking a soft checkpoint. All `logprimary`, `logfilsiz` and `softmax` are database configuration parameters.

When page cleaners are activated, they build a list of dirty pages. In this list, pages are ordered by the time a page was first changed. Notice that these pages may not neccesarily be the least recently used. Once they have written all these pages back to disk, they will become inactive again.

The number of page cleaners activated is determined by the database configuration parameter, `num_iocleaners`. By default, it is set to one. Changing this value may have a big impact if the database is stored across several disks. A good value for this parameter, in a transaction environment, is the number of physical devices used by the database.

In a query-only database, I/O cleaners have little impact. The only time that they will come into play is if a large temporary tables needs to be created.

These following two data elements help to tune the activity of page cleaners:

- Buffer Pool Threshold Cleaners Triggered

  Indicates the number of times a page cleaner was invoked because the buffer pool has reached the dirty page threshold set by `maxchngpgs`.

  If cleaners are triggered often, then you might consider either increasing the `maxchngpgs` or the buffer pool settings.

- Buffer Pool Log Space Cleaners Triggered

  Number of times a page cleaner was invoked because the size of the log that would need to be read to recover the database is greater than `logprimary * logfilsiz * softmax`. Logging is discussed in 3.2, “Logging” on page 32.

  If the number of cleaners triggered due to log space recovery is high, the `softmax` parameter could be increased. Notice that this will impact the time required to recover the database in case it crashes.

3.1.1.6 Summary

When tuning the buffer pool, the goal is to reduce the number of I/Os. To reduce the number of I/O operations, the data pages must be placed in the buffer pool before they are needed. Applications will wait on synchronous I/Os, so buffer pool operations performed by I/O cleaners (writing pages from the buffer pool to disk), should be performed asynchronously whenever possible. The configuration parameters involved are:

- `buffpage`

  Size of the buffer pool.

- Prefetch size of the tablespaces

  Max number of pages that the I/O servers can prefetch. The prefetch size of a tablespace is determined when the tablespace is created. It can be modified through the `alter tablespace` command. Sequential prefetching can be turned off using the `seqdetect` configuration parameter.

- `num_iocleaners`

  Maximum number of I/O cleaners.

- `maxchngpgs`

  Percentage of “dirty pages” that trigger the I/O cleaners.

To detect if the buffer pool is undersized, the best way is to check the number of times the page cleaners were triggered. For query-only environments, no page cleaners will be triggered as no pages are modified. Other clues to detect that the configured buffer pool size is small are:

- I/O servers not prefetching as many pages as expected
- A low Buffer pool-sync hit ratio
- A low Index pool-sync hit ratio

A first approach when running the buffer pool size is to increase it until a good index pool sync hit ratio is achieved (over 80 percent, for example). If more memory is available, try to increase it until the buffer pool sync hit ratio goes over 80 percent.

Notice that this ratio is affected by the following factors:

1. Prefetchers

   If no prefetching is performed and the data pages are not already in the buffer pool, the buffer pool sync ratio and the response time will be severely affected when scanning large amounts of data.

2. The Prefetch Size of the tablespaces

If the Prefetch Size of a table space is increased, be careful to avoid disk contention. If I/O servers are prefetching too many pages, contention may arise. This is detected though the use of operating system tools, and confirmed when non-steady values of the synchronous I/O read time are measured.

3. Table structure

   Table reorganization only applies if data is being accessed through an index. Choose the most frequently used index and cluster the data according to this index. Table reorgs are detected when the response time for queries accessing a table is lower than expected, and can be confirmed when reorgchk shows low cluster ratio for the index.

3.1.2 Database Heap

The Database Heap maintains control blocks for tables, views and tablespaces, maintaining a descriptor for each page in the buffer pool. It also holds the Log Buffer and the Catalog Cache. The Database Heap is allocated within the Database Global Memory Area.

The Database Heap maximum size is determined by the database configuration parameter, dbheap. This parameter should be set to a value so that the Log Buffer, Catalog Cache and buffer pool page descriptors fit into the database heap. The area required by descriptors is approximately 1/30 the size of the buffer pool. The rest of the space is allocated when needed. The default maximum size of the database heap is 1200 4KB pages for AIX servers, 600 4KB pages for OS/2 and NT servers with remote clients and 300 4KB pages for OS/2 and NT servers with local clients only.

Increasing the dbheap parameter has an impact on availability, not performance. If the heap is exhausted, connections will receive errors and may have to roll back their transactions. If the 'Maximum Database Heap Allocation' data element gets close to the value set by dbheap, the dbheap value should be increased.

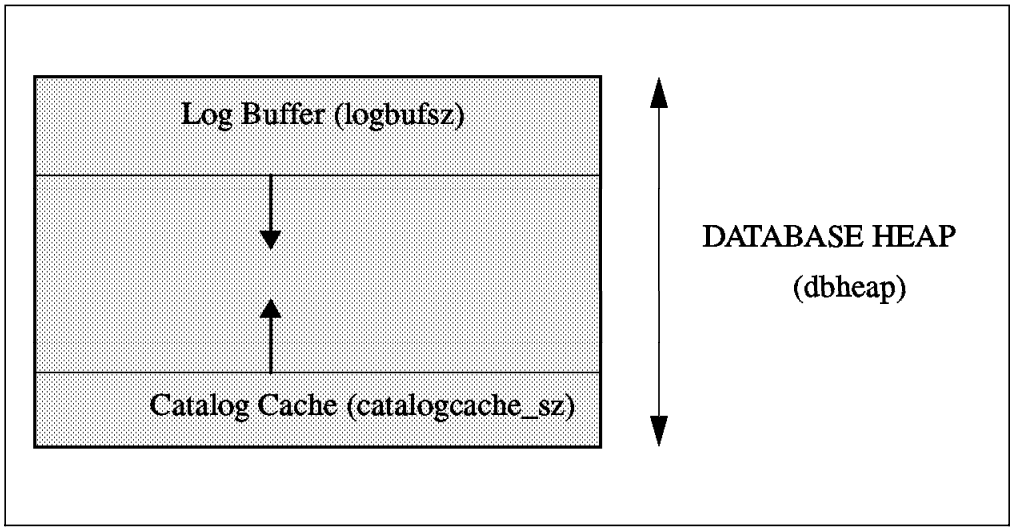


Figure 10. The Database Heap

Figure 10 shows that the Log Buffer and the Catalog Cache are allocated from the Database Heap. Both Log Buffer and Catalog Cache use only the memory

they need, and their maximum size is limited by the logbufsz and catalogcache_sz configuration parameters.

3.1.2.1 Log Buffer Size

The size of the log buffer is limited by the logbufsz database configuration parameter. By default, the maximum size is set to 8 4KB pages for all servers. This memory is allocated as required and released when not in use. The memory is allocated from the Database Heap.

The log buffer should have, at least, the space required by a transaction. If the size of the buffer is big enough, there will be no need to perform an I/O operation (write to the log file) until the transaction is committed. If the size of the buffer is too small, it will write to the log file when the buffer fills up. If commits are being grouped, the size of the log buffer should be multiplied by the number of commits being grouped. Grouping commits will result in less I/O activity.

To assist in the log space used by a specific transaction, the following data element is provided:

- Unit of Work Log Space Used

  Amount of log space used by the UOW being monitored.

Figure 11 shows the influence of the size of the log buffer in the response time of an SQL update statement. The statement updated more than 6000 rows, and required 2,163,779 bytes of log space. This log space was measured by the Unit of Work Log Space Used data element. The number of log pages written was 161, which is above the maximum value of 128 pages(logbufsz maximum configurable value). The response time decreased by almost 6 percent.

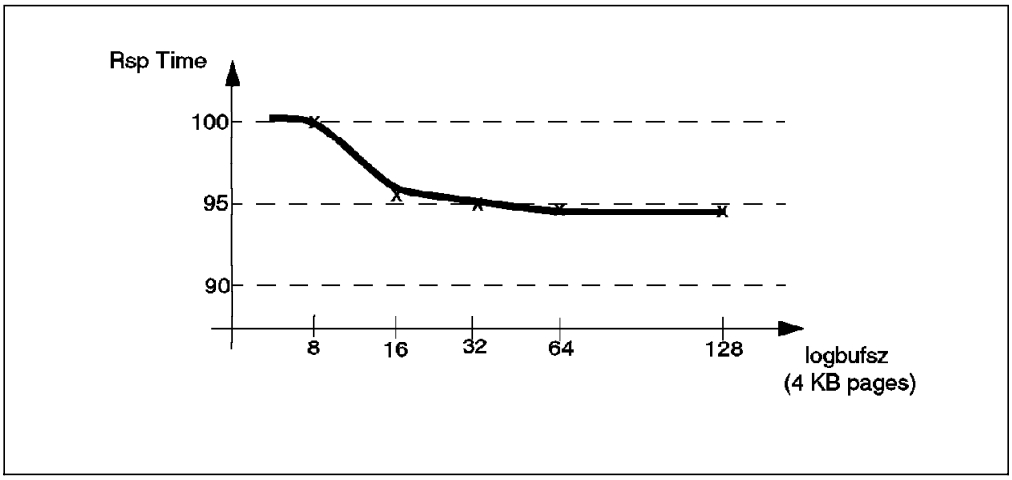


Figure 11. Response Time and Log Buffer Size

Logging is further discussed in 3.2, “Logging” on page 32.

3.1.2.2 Catalog Cache Size

The Catalog Cache keeps references to the catalog in a memory area. It is used to avoid disk reads when binding SQL statements, static or dynamic.

This cache is only allocated when required. Its default maximum size is 64 4KB pages for AIX servers, 32 4KB pages for OS/2 and NT servers with remote clients and 16 pages for OS/2 servers with local clients. It is allocated within the

database heap and its size is determined by the catalogcache_sz database configuration parameter.

The purpose of this cache is to avoid disk reads when compiling SQL statements or using dynamic SQL. It is not used in regular 'transaction static SQL' environments. It stores table descriptors for each table, view or alias referenced. A table descriptor is inserted by a transaction so that following transactions will find the table descriptor in the cache the next time the table is referenced. Notice that a DDL statement against a table will cause the table descriptor of this table to be flushed from the cache. The cache is deallocated when all the applications disconnect from the database.

The following data elements monitor the use of the catalog cache:

- Catalog cache inserts

  Number table descriptors inserted into the cache.

- Catalog cache lookups

  Number of times statements are found in the catalog cache for a table descriptor.

The ratio calculated dividing the number of 'catalog_cache_inserts' by the number of 'catalog_cache_lookups' should not exceed 0.2. This will indicate that only one out of five catalog cache lookups have required a catalog cache inserts. Some inserts cannot be avoided, but the goal is to avoid an undersized cache that inhibits cache hits or that is unable to provide space for complex SQL statement compilations.

The following data elements indicate insufficient sizing of the catalog cache or the database heap:

- Catalog cache overflows

  Number of times a catalog cache insert failed caused by a catalog cache full condition. If there are several overflows, the size of the catalog cache should be increased.

- Catalog cache heap full

  Number of times a catalog cache insert failed caused by a database heap full condition. If the number is more than a few, the database heap should be increased.

When sizing the catalog cache, the goal of the database administrator is to set its size so all table and view descriptors could fit in the cache. This may not always be possible due to memory restrictions. For our database, which stores sixty-three tables, we tested with different sizes of the cache. We created a loop of selects that referenced all the tables of the database, and executed this loop several times from two different applications. The results are plotted in Figure 12 on page 27. For our database, the optimal size of the catalog cache is shown to be 32 4KB pages.

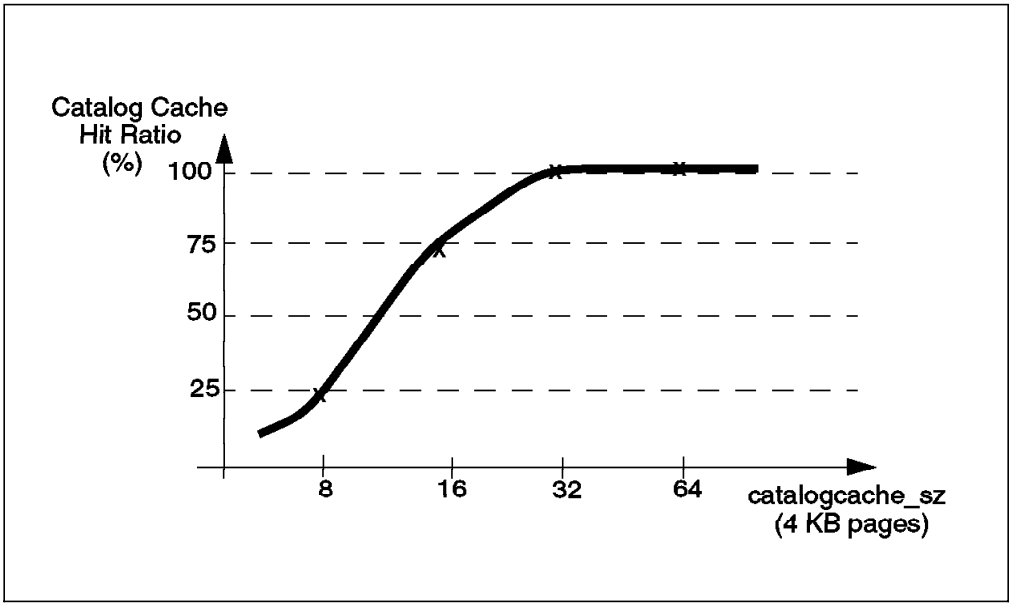


Figure 12. Catalog Cache Hit Ratio/Catalog Cache Size

To test the effect of the catalog cache on the response time, we executed a query against a table. Two sets of runs of this query were made. In one of the sets, the descriptor of the referenced table was in the cache, and in the second run, it was not. Results are shown in Table 2.

	Referenced table not in Catalog Cache	**Referenced table in Catalog Cache**
Response Time (%)	100	97.21
Catalog cache lookups	1	1
Catalog cache inserts	1	0
Catalog cache size	4 Pages	64 Pages

Table 2. Response Time and Catalog Cache

3.1.2.3 Summary

The database heap, if properly sized, does not have as big an impact in the performance of the database as the buffer pool has. But if not properly sized, errors will appear and the performance of dynamic SQL statements will be affected. It contains room for the log buffers and for the catalog cache.

The database configuration parameters involved when dealing with this heap are:

`dbheap` Size of the database heap.

`logbufsz` Size of the log buffers.

`mincommit` Number of commits to group together. (Refer to 3.2.2, “Grouping Commits” on page 33)

`catalogcache_sz` Size of the catalog cache.

To detect if the database heap is undersized, check the ‘Maximum Database Heap Allocation’ water mark. If this gets close to the `dbheap` value, then the

database heap should be increased. The parameter, logbufsz, is discussed further in 3.2.1, “Sizing Log Buffers” on page 32 and mincommit in 3.2.2, “Grouping Commits” on page 33.

Catalog cache problems are detected when:

- Catalog cache overflows appear

  This points to an insufficient value for the catalogcache_sz.
- Catalog cache heap full conditions appear

  This data element points to an undersized dbheap.

3.1.3 Locks

An application may have to wait if the data it needs to access is locked by another application, or it will wait if it needs to obtain a lock and no locks are available. The memory area reserved for locks is called the Lock List. The Lock List is allocated within the Database Global Memory. Locks are used to guarantee data integrity. Number of locks used will depend upon:

- The number of concurrent applications
- Isolation levels used by applications
- The number of rows affected in transactions

3.1.3.1 Available Locks

The number of locks available for a database is determined by the locklist database configuration parameter. This parameter sets the amount of memory assigned to lock lists. By default, it is set to 100 4KB pages for AIX servers, 50 4KB pages for OS/2 and NT servers with remote clients and 25 pages for OS/2 and NT servers with local clients. Each lock requires either:

- 64 bytes if the object being locked has no other locks on it
- 32 bytes if the object already has a lock held on it

Memory is allocated when the database is activated. The following data elements can be monitored to help check that the lock list is correctly sized:

- Total Lock List Memory in Use

  Amount of memory, in bytes, that is in use within the lock list. This data element should be sampled, and an average value must be obtained. The average value of the lock list memory in use should be less than 90 percent of the memory set by the locklist configuration parameter.
- Locks Held

  The number of locks currently held. When this data element is measured at database level, it can provide guidance on the size of the lock list. Again, this element should be sampled and an average value obtained. From this average, the average use of the lock list can be obtained using the following:

$$\text{LockListUsed} = \frac{\text{LocksHeld} \times 32}{\text{locklist} \times 4096} \times 100$$

3.1.3.2 Avoiding Lock Escalations

Lock escalations occur when a unit of work requires more locks than the database server is configured to assign. This will happen when:

- A single unit of work (UOW) tries to monopolize a big chunk of the lock list. The server limits the maximum percentage of locks from the lock list that a UOW can hold. It does this to ensure that other units of work will have locks available. The limit is imposed through the maxlocks database configuration parameter. By default, it is set to 10 percent for AIX servers and to 22 percent for OS/2 and NT servers.
- The lock list is undersized. If a UOW requests more locks than those available in the lock list, a lock escalation will occur.

Escalations reduce the concurrency of the database and can lead to deadlocks. The maxlocks parameter can be tuned using the following data elements:

- Locks Held

  The number of locks currently held. When this data element is measured at the application level, it provides the number of locks that are being held by the application. This value has to be compared to maximum number of locks that the database will allow to be held by an application.

- Maximum Number of Locks Held

  The maximum number of locks held by the transaction being monitored.

- Lock Escalations

  The number of times that locks have been escalated from row locks to a table lock. It can be estimated at the database level.

$$\text{LocksHeld} < \frac{(\text{locklist} \times 4096 / 32)}{\text{maxlocks} / 32} \times 100$$

Concurrence-related problems can be detected through the monitoring of the following data elements:

- Current Applications Waiting On Locks

  Number of applications waiting on a lock.

- Applications Connected Currently

  Indicates the number of applications that are currently connected to the database.

- Lock Waits

  Number of times that applications waited for locks.

When tuning locks, the goal is always to increase concurrency. Using different isolation levels, and thus, possibly reducing the number of locks required by a unit of work, is one of your options. The size of the locklist or the maximum percentage of the locklist held by a unit of work can also be increased.

3.1.3.3 Deadlocks

Deadlocks appear when two applications are waiting for resources that are being locked by each other. The database manager will roll back one of them to resolve the deadlock. Deadlocks can be minimized by:

- Using correct isolation levels
- Avoiding lock escalation

Deadlocks can be detected through these data elements:

- Deadlocks Detected

  Total number of deadlocks that have occurred. The database manager checks for deadlocks at the frequency determined by the `dlchktime` configuration parameter. Its default value is 10000 milliseconds.

- Number of Locks Timeouts

  Number of times that a request for a lock timed out. Applications will wait on locks according to the `locktimeout` database configuration parameter. By default, this parameter is set to -1, which means that no lock timeout check is made and the application will wait forever. A value of 30 seconds is reasonable for an OLTP environment. For a batch environment, 60 seconds is a good initial value.

The interval for deadlock checking, `dlchktime`, should be set to be less than the `locktimeout` interval. Doing this, deadlocks are eliminated by rolling back one of the conflicting applications, giving a chance to the non-rolled-back application that is waiting for locks to complete its work.

3.1.3.4 Summary

The lock list is the memory area used by the database to store locks. Locks affect the concurrence of the database. An improper size of the lock list may cause:

- Lock escalations which reduce concurrency and increase the chance of deadlocks.
- Applications waiting on locks.
- Timeouts for applications that have been waiting on locks.

The database configuration parameters involved when dealing with the lock list are:

`locklist`	Maximum size of the memory area reserved for locks.
`maxlocks`	Maximum percentage of locks from the lock list that a UOW can hold.
`dlchktime`	Interval between deadlock detections.
`locktimeout`	Time that an application will wait to obtain a lock before it times out.

To detect if the lock list is undersized, you can check the ‘Lock Waits’ and the ‘Lock Escalations’ data elements. Having no lock waits and no lock escalations points to a well-sized lock list.

If escalations appear, check for possible “greedy” applications and monitor them through the ‘Locks Held’ and the ‘Maximum Number of Locks Held’ data elements. A possible solution is to change the isolation level of the application,

increase the maxlocks configuration parameter or increase the size of the Lock List.

Deadlocks and lock timeouts point to problems with escalations or with isolation levels. They can be detected through the 'Deadlocks Detected' and the 'Number of Locks Timeouts' data elements. If either of these two data elements collects a significant value, correct your database configuration, increasing the lock list, or change the isolation level of the applications.

3.1.4 Utilities and Recovery

The memory area used by the LOAD, BACKUP and RESTORE utilities is called the utility heap. The utility heap is allocated within the Database Global Memory area. The utility heap includes both backup and restore buffers. It is only allocated when needed. The maximum size of this heap is, by default, 5000 4KB pages for all AIX, OS/2 or NT servers. Notice that the number of I/O servers (async servers) available will also impact the performance of the backup and restore utilities. Figure 13 shows the backup and the restore buffers allocated within the utility heap. The amount of memory allocated for each utility will depend on the size of the buffers and the number of them being used.

Utilities do not use the buffer pool. If more space is needed for the utility heap, the size of the buffer pool can temporarily be reduced. Notice that the buffer pool size cannot be changed while the database is active.

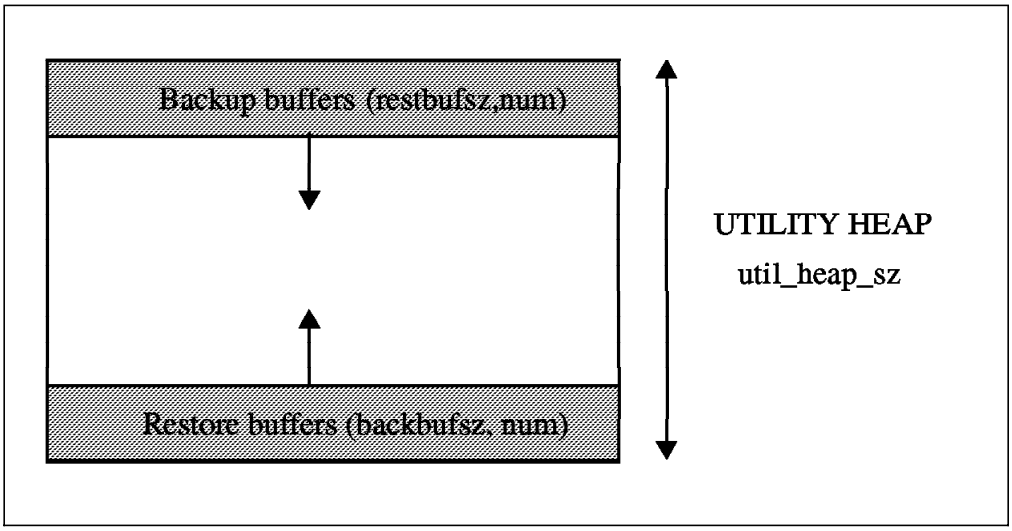


Figure 13. The Utility Heap

3.1.4.1 Backup

The number of buffers and their size can be defined when the backup command is executed. If not, the buffer size is set to the backbufsz database configuration parameter, which has the default value of 1024 4KB pages.

For better performance, the number of buffers to be used during a backup should be set to be at least twice the number of target physical devices where the backup is being sent. This is due to the fact that devices are slower than the ability of the database to write to a backup buffer. By doing so, the database can write to a second backup buffer while the first backup buffer is being written to the device.

3.1.4.2 Restore

The number of buffers and their size are usually determined when the restore command is executed. If not, the buffer size is set to the restbufsz database configuration parameter. Its default value is 1024 4KB pages.

The number of buffers to be used during a restore should be set to be at least twice the number of source physical devices from where the backup is being restored for the same reasons described in 3.1.4.1, “Backup” on page 31.

3.2 Logging

From a performance point of view, two issues apply when discussing logging:

1. The overhead of allocating primary or secondary log files.
   - If using circular logging, all primary log files are pre-allocated and are reused. Secondary log files are allocated when required and then deallocated when no longer required. This creates allocated disk space that can be avoided. To do so, the number and size of the primary log files should be set to handle the daily transaction needs, avoiding secondary log file allocation
   - If using archival logging, primary log files are not reused. They are retained, and new primary log files are allocated as the old ones are archived. No secondary log files are used with archival logging. The primary log file size should be set, if possible, to a high enough value to minimize the overhead of allocating new primary log files.
2. Log activity, including when logs are written to disk, may have an impact on the response time of applications. Writing to logs is always necessary, but the I/O activity can be reduced by choosing an appropiate size for the log buffer and grouping commits.

3.2.1 Sizing Log Buffers

The size of the log buffer is limited by the logbufsz database configuration parameter and by the dbheap configuration parameter, as shown in Figure 10 on page 24. By default, the maximum size is set to 8 4KB pages. This memory is allocated when needed and freed when not in use. The memory is allocated from, and so limited to, the database heap.

The log buffer should have, at least, the space required by an average transaction. When the buffer fills, the log data is written to disk. If commits are being grouped, the size of the log buffer should be multiplied by the number of commits being grouped.

The space required by a transaction can be estimated using the following data element:

- Unit of Work Log Space Used

  Amount of log space, in bytes, used in the current UOW of the monitored application. Values should be obtained for a representative transaction.

3.2.2 Grouping Commits

Commits are grouped according to the mincommit database configuration parameter. Setting this parameter to a different value than its default value (1) will attempt to group commits. If the number of transactions per second being executed are less than the mincommit value, transactions will be committed every second.

The parameter mincommit should be adjusted to the number of transactions per second at the peak hour. To obtain the number of transactions executed, the following data elements must be measured during the peak hour interval:

1. Commit Statements Attempted

   Total number of SQL commit statements that have been attempted.

2. Rollback Statements Attempted

   Total number of SQL rollback statements that have been attempted.

Add together the values obtained from these two measurements and then divide by the number of seconds in the measured interval. This will tell you the number of transactions per second.

3.2.3 Sizing Log Files

The following two database configuration parameters determine the appropiate size for the primary log files:

- logprimary

  Sets the number of primary log files to be used by the database. By default, the database will use three log files.

- logfilsiz

  Sets the size of these files. Its default value is 1000 4KB pages for AIX servers and 250 4KB pages for OS/2 and NT servers.

When circular logging is being used, the number of secondary log files allocated is given by the following data element:

- Secondary Logs Allocated Currently. If not zero, the next two data elements show if the sizes and number of primary log files are much out of target:
  - Maximum Secondary Log Space Used

    Maximum amount of secondary log space used in bytes.

  - Maximum Total Log Space Used

    Maximum amount of total log space used in bytes. This should be compared to the amount of space allocated to primary log files.

3.2.4 Summary

To log efficiently, the size of the log buffer must be large enough to meet the log space requirements of a transaction. If commits are being grouped, multiply the size of log buffers by the number of transactions being committed. There are two database configuration parameters that regulate this:

- mincommit

  Number of transactions being grouped.
- logbufsz

Size of the log buffers.

Apart from buffers, the size and number of log files are set by the logprimary and the logfilsiz configuration parameters. Circular or archived logging is determined by the logretain configuration parameter.

3.3 Heaps used by Agents

Each agent has a private memory area. This private memory area contains the following heaps and stacks:

1. Application Heap. This heap is used for executing all SQL statements.
2. Sort Heap. Used for sorting tables.
3. Statement Heap. Used for compiling SQL statements.
4. Agent Stack. Controls the amount of memory used by each agent.
5. Statistics Heap. Heap used for gathering statistics data.
6. DRDA Heap.
7. UDF Memory. Heap used to exchange data between UDFs and the database.
8. Query Heap. Heap used to process requests/replies to/from local applications.
9. Client I/O Block. Memory area to process requests/replies from remote applications.

Notice that a private memory area is allocated for each agent (active or idle) in the machine where the database server is running. Every concurrent application (local or remote) will be served by a separate agent.

3.3.1 Application Heap

One application heap is allocated with each connection being made. By default, the maximum size of the application heap is 128 4KB pages (AIX, OS/2 or NT servers). Only the minimum amount of memory required is allocated when the application connects to the database. More memory is allocated as required, until the maximum size of the application heap is reached. It is configured by the applheapsz parameter. The application heap contains the package cache and memory for other internal uses.

3.3.1.1 Package Cache

Package Cache is used to maintain the most-frequently accessed sections of the package. The maximum size of this cache is, by default, 36 4KB pages (AIX, OS/2 or NT servers), and it is configured by the pckcachesz parameter. It is allocated from the application heap. The compiled SQL statements (packages) are kept in the cache until one of the following occurs:

- The package is purged from the cache
- The cache fills up
- The application disconnects from the database

Sections of the package are purged from the cache when certain DDL statements are executed. Invalid sections of the package will be recompiled and will be inserted in the cache the next time they are used. Three data elements help to monitor the package cache of an agent:

- Package Cache Inserts

  Number of times a package section was not in the cache and had to be inserted.

- Package Cache Lookups

  Number of times an application looked for a package section in the package cache.

- Data Definition Language SQL Statements

  This element can be measured at database or application level.

Success of this cache can be measured, at either the database or application level, using the following ratio:

$$\text{PkgCacheHitRatio} = \frac{\text{PackageCacheLookups} - \text{PackageCacheInserts}}{\text{PackageCacheLookups}} \times 100$$

If no DDL statements are being executed, a low hit ratio points to a small package cache or to a small application heap size. If the ratio is over 80 percent, the cache is considered to be working well.

The Package Cache has a big impact in application performance both for static and dynamic applications.

3.3.2 Sorts

The Sort Heap is used to sort data-index or data pages. Its maximum size is determined by the sortheap database configuration parameter. It's only allocated when required, and its default maximum value is 256 4KB pages for AIX, OS/2 and NT servers. The sortheap sets the maximum amount of memory available for a sort. When the sorting is done, the memory used is released.

The size of the heap should be increased when large sorts are frequently used. The larger the table, the higher the sortheap value required. At instance level, the total amount of memory to be used by sorts is limited by the sheapthres configuration parameter. This parameter is set to 4096 4KB pages for AIX servers and to 2048 4KB pages for OS/2 and NT servers

When the amount of memory required by all sorts (at the instance level) reaches sheapthres, new sort requests will not be denied; sheapthres is not a hard limit. New sort requests will be accepted, but the database manager will not use piped sorts to process the incoming sort requests.

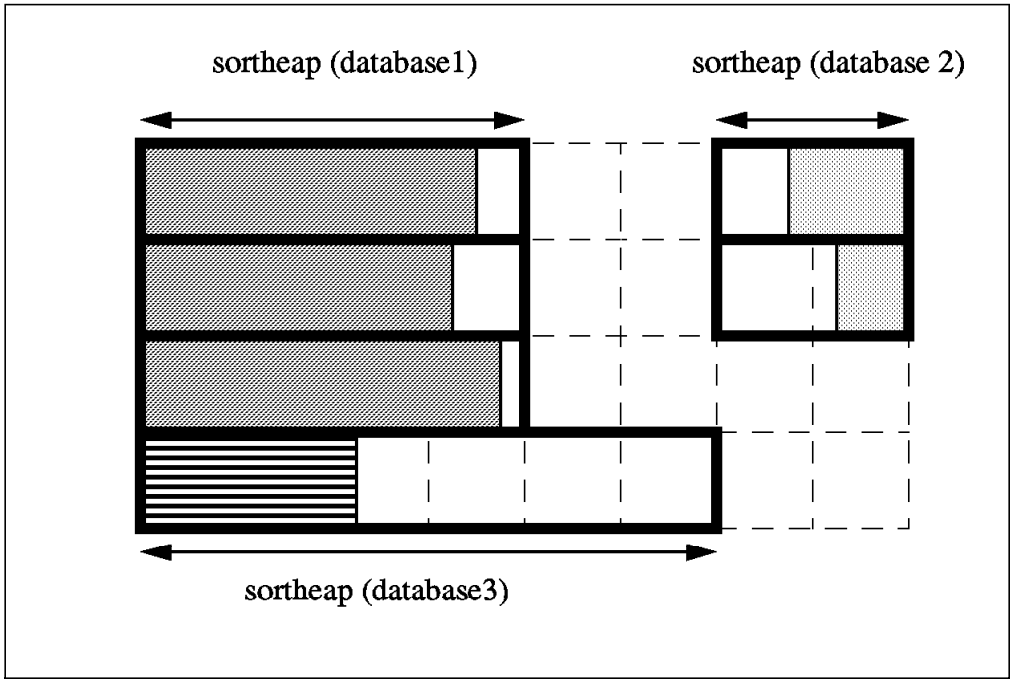


Figure 14. Sort Heap and Sort Heap Threshold

In Figure 14 the dashed area represents the sort heap threshold, while the bold lines represent the sortheaps of each database. The figure shows three sorts being executed against database 1, two against DATABASE 2 and one against database 3. Notice that each database may have set different values for sortheap. All agents performing sorts are limited by their own sortheap configuration parameters. Global resources for sorting within an instance are limited by the sheapthres configuration parameter.

Piped sorts provide a performance benefit. A sort is piped if the rows requested by the application fit into the sort heap. A non-piped sort will use temporary tables to perform the sorting. This is done by dividing the sort in several passes where each pass sorts a subset of the entire set of rows being sorted. After all passes are completed, a merge between all subsets is performed. These temporary tables are stored in the buffer pool, so they can be paged to disk by the page cleaners. Piped sorts have these performance benefits over non-piped sorts:

1. Piped sorts may reduce disk I/O activity.
2. Sorts are performed in one phase.

The data elements associated with the sheapthres are:

- Piped Sorts Requested

  Number of piped sorts, at instance level, that have been requested.

- Piped Sorts Accepted

  Number of piped sorts, at instance level, that have been accepted.

- Post Threshold Sorts

  Number of sorts that have requested heaps after the sheapthres has been reached.

- Total Sort Heap Allocated

Total number of allocated pages for all sorts. This should be measured at instance level and compared to the sort heap threshold.

The sort heap threshold should be increased when:

1. The number of post threshold sorts is significant when compared to the total number of sorts.
2. The difference between the number of piped sorts requested and the number of piped sorts accepted is high. Notice that if there are no post threshold sorts, but the difference of piped sorts requested and accepted is high, then the sortheap is the cause and its value should be increased.
3. The total sort heap allocated gets near the value set for the threshold.

Figure 15 shows the response time of a query using different sort heap sizes. In no iteration was the sort heap threshold set by sheapthres surpassed. The query sorted a fairly large amount of data. Notice that the response time improved when the size of the sortheap was increased. The big increase in response time is obtained when the sort did not overflow the sort heap. For small values of the sortheap configuration parameter, the sort request overflowed the heap, requiring the use of the buffer pool and the use of several sort phases and a merge phase.

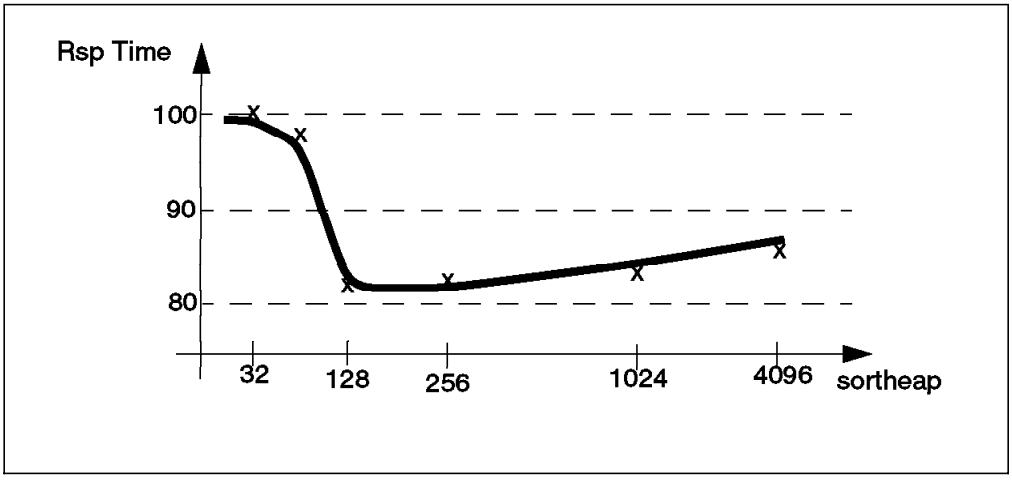


Figure 15. Sort Heap and Response Time

Another conclusion obtained from the figure is that using a sortheap size larger than needed does not benefit the response time of the query. The goal of the database administrator is to find the spot where as many sort overflows as possible are avoided.

The sort heap is monitored through the following data elements, which can be measured by the DB2 performance monitor or the snapshot monitor:

- Sort Overflows

  Number of sorts that overflowed the sort heap, requiring space in the buffer pool to complete. A high number of sort overflows point to the Sort Heap being too small.

- Active Sorts

  Number of sorts in the database that currently have a sort heap allocated.

- Total Sort Heap Allocated

Measured at database level, total number of allocated pages for all sorts.

The average size of the sort heap being used by an application can be obtained by sampling the number of active sorts and the total sort heap allocated. This value can then be compared to the sortheap configuration parameter.

3.3.3 Monitoring Agents

An agent is a separate process or thread that carries all requests made by an application. Every application will have its own agent which is dynamically created when required. The database will assign one of the idle agents to attend to the requests of each application.

The number of agents is limited by the lower of the following:

1. Maximum number of applications that can be connected to a single database. This number is set by the maxappls configuration parameter. By default, it is set to 40 for AIX servers, 20 for OS/2 and NT servers with local and remote clients, and 10 for OS/2 and NT servers with local clients only.
2. The maximum number of agents that can be concurrently executing a transaction. This is configured through the maxcagents configuration parameter. Its default value is the value set by the maxagents parameter.
3. The maximum number of agents for all databases created under a single instance. This is set by the maxagents configuration parameter. By default, it is set to 200 for AIX, OS/2 and NT servers.

Appropriate values for these configuration parameters can be set by:

1. maxagents

   These data elements apply to:

   - Local Connections

     Total number of local clients connected to any database of the instance.

   - Remote Connections to Database Manager

     Total number of remote clients connected to any database of the instance.

   - Maximum Number of Agents Registered

     Maximum number of agents ever connected to any database of this instance since the instance was started.

   The values of maxagents should be set to be greater than the sum of local and remote connections. It should also be increased if the maximum number of agents registered gets close to the current value of maxagents.

2. maxappls

   - Applications Connected Currently: Number of applications connected to the database being monitored.

3. maxcagents

   - Maximum number of Agents Waiting

     Highest number of agents that have been waiting, at the same time, for their transactions to be executed because the maxcagents limit has been reached. This highest number is recorded since the instance was started.

- Maximum Number of Agents Registered

  Notice that this is the high-water mark for the number of agents connected at the same time.

3.3.3.1 Other performance issues

To avoid the overhead of creating new agents when a connection is requested, the database manager can keep a number of idle agents. The number of idle agents not assigned to any client is set by the max_idleagents configuration parameter. By default, it is set to three for AIX, OS/2 and NT servers. When an agent is released by the client, the database manager will not terminate the agent process if the number of idle agents available is less than max_idleagents.

Agent priority can be set by the agentpri configuration parameter. By default it is set to -1, meaning that all database processes and agents should be treated by the operating system as any other process. Performance can be increased setting a higher priority. Care should be taken when setting this parameter because all other user processes, including those not related to DB2, will be affected.

3.3.4 Client Applications

Clients and applications are used to represent the same concept. A client is said to be local if it is being executed on the same machine as the database server. A remote client, is the client being executed on another machine. To attend the requests of a local client, a query heap is allocated within the agent private memory. If the request comes from a remote client, a client I/O block is allocated in the agent private memory. So, for a given client, only one of these structures will be allocated; either a query heap or a client I/O block.

3.3.4.1 Local Clients - Query Heap

This heap is closely related to the application support layer heap. This heap is used to store each query in the agent's memory. It provides room to store the following:

1. The statement text
2. Package name and creator
3. Input and output SQLDA
4. The SQLCA

The initial size of this heap is determined by the size of the application support layer heap. Its maximum size is set by the query_heap_sz configuration parameter. The value for the query_heap_sz parameter should be set so it fits the requests/replies coming from the application support layer and leaves enough room for blocking cursors.

The default setting of the query_heap_sz is 1000 4KB pages for AIX, OS/2 and NT servers. Notice that the default size for the application support layer heap is 15 4KB pages.

3.3.4.2 Remote Clients - Client I/O Block

The client I/O block of memory serves the request/replies from and to remote applications. The default size of this I/O block is 32KB for AIX, OS/2 or NT servers. The server will set the I/O block value to the value specified by the rqrioblk or dos_rqrioblk at the client when the connection is made. The dos_rqrioblk parameter applies only to DOS clients, while the rqrioblk parameter applies to non-DOS clients.

3.3.5 Other Heaps

Other heaps are also allocated in the agent private memory. They have little or medium impact in performance. They are described here for completeness purposes.

1. Statement heap

   The statement heap is used for the compilation of SQL statements. For dynamic SQL, it is used during execution of the application, while for static SQL it is only used during binding. Its maximum size is set by the stmtheap configuration parameter. By default, it is set to 2048 4KB pages for AIX, OS/2 or NT servers.

2. Statistics heap

   This heap is used by the runstats command to collect statistics. It is allocated when runstats is executed and freed when no runstats are in use. Its maximum size is set ny the stat_heap_sz configuration parameter. By default, it is set to 4384 4KB pages for AIX, OS/2 or NT servers.

3. Agent stack

   The stack is the amount of memory allocated by the operating system to each agent. It is configured through the agent_stack_sz configuration parameter, and its default size is 64 4KB pages for AIX and OS/2 servers and 16 4KB pages for NT servers.

4. DRDA heap

   The DRDA heap is used by DDCS and the DRDA application server. The DRDA application server will allocate a heap for each DRDA application request making a connection to the database. DDCS requestor will allocate a heap when it connects to a DRDA application server. By default, the size of this heap is set to 128 4KB pages for AIX, OS/2 and NT servers through the drda_heap_sz configuration parameter.

5. UDF heap

   This heap is used to exchange data between the user-defined functions used by the application and the database. It is configured through the udf_mem_sz parameter. Its default value is 256 4KB pages for AIX, OS/2 and NT servers. If the application being served by the agent does not use user-defined functions, then this heap is not allocated.

3.3.6 Summary

The agent private memory area contains many heaps and stacks, but the two important heaps dealing with agents are the application heap (which includes the package cache) and the sort heap.

The success of the package cache is measured through the 'package cache hit ratio' data element. The size of the package cache is limited by both of these configuration parameters:

- pckcachesz

  Maximum size of the package cache.
- applheapsz

  Maximum size of the application heap.

When sorting data, the goal is to perform as many piped sorts as possible. The amount of memory available for the sort is limited by the sortheap configuration parameter. The overall sort memory resources, within an instance, are also limited by the sort heap threshold. This is determined by the instance configuration parameter, sheapthres. This threshold is a "soft" limit. New sort requests will be accepted if the threshold is surpassed. But the database manager will not perform piped sorts.

Problems with the sort heap threshold are detected when:

- The post threshold sorts data element is not zero.
- The difference between the number of piped sorts accepted and the number of piped sorts requested data elements is too high.
- The total sort heap allocated data element (measured by a database manger snapshot) gets close to the amount of memory limited by sheapthres.

Sort heap problems are detected when:

- The sort overflows data element is not zero.
- Piped sorts are rejected, but there are no post-threshold sorts.

3.4 Heaps Used by Applications

Applications use one of these heaps to exchange information between the application and the agent. The heap also is used for row blocking, which retrieves a block of rows in a single operation. These rows are stored in a memory area of one of the heaps, so each FETCH request will get the next row from this memory area. The memory area is allocated when the application opens a cursor. Blocking will be used depending on the options specified when precompiling and binding the application.

The memory area used is determined by the location of the application, whether is it local or remote:

1. If the application is a local application, the application support layer heap is used.

   The application support layer heap is configured through the aslheapsz database configuration parameter. By default, this is 15 4KB pages. This value should be sized to contain an average request/reply between the application and its agent. Its size should be increased if queries retrieving large amounts of data are used.

2. If the application is a remote application, the client I/O block is used.

   This memory is allocated in the client. The default size of this I/O block is 32KB for non DOS/Windows clients and 4KB for DOS/Windows clients. It is configured by the rqrioblk or by the dos_rqrioblk configuration

parameter. It is independent of the transport protocol used to connect the client and the server.

Notice that an I/O block will be opened for every client connection to a database. If a client is concurrently connected to several databases, several I/O blocks will be opened.

3.5 Binds and Tuning Parameters

There are three bind-related factors that impact the performance of SQL statements, both static and dynamic. These three factors are:

- The Isolation Level

  The isolation level affects the performance of the application as CPU and database resources are used to obtain and free locks.

- Blocking

  Blocks of data can be retrieved in a single operation, improving the performance of a query.

- Optimization Class

  The optimization class is used by the optimizer to determine the access plan.

3.5.1 Isolation Level

The isolation level determines how data is locked from other agents while being accessed. Every package has its isolation level. The isolation level is specified when the application is precompiled (`prep` command) or bound (`bind` command). These four possible isolation levels:

1. Repeatable Read

   The application will lock all rows referenced within a unit of work. The optimizer will determine, based on the `locklist` and `maxlocks` configuration parameters, whether to lock the table for the access plan instead of locking individual rows.
2. Read Stability

   The application will lock all the rows retrieved within a unit of work.
3. Cursor Stability

   The application will lock the row where the cursor is currently positioned.
4. Uncommitted Read

   No locks are acquired. The application does not even lock the row that it is reading.

From a performance point of view, this means that if a query scans a 75000-row table and only retrieves 500 though a predicate, it will lock all 75000 rows if repeatable read is used (either holding a lock against each individual row or locking the whole table), 500 rows if read stability is used, one row at a time if cursor stability is used, and no rows if uncommitted read is used.

3.5.2 Blocking

Cursor blocking is determined when applications are precompiled or bound. Blocking is specified through the BLOCKING option of these commands. BLOCKING options are:

- UNAMBIG

  All cursors, not explicitly defined FOR UPDATE, are blocked.

- ALL

  Ambiguous cursors are blocked.

- NO

  Cursors are not blocked.

If no BLOCKING option is specified, the default row blocking is UNAMBIG, except for the command line processor and CLI, where, by default, blocking is ALL.

When blocking is used, the number of rows returned (per block) to the application is:

- For local applications

  The size of the application support layer heap (in bytes) divided by the output row length (in bytes).

- For remote applications

  The size of the client I/O block (in bytes) divided by the output row length (in bytes).

Notice that the number of rows per block returned to the application may be conditioned by the OPTIMIZE FOR n ROWS clause of a SELECT SQL statement (but notice that the OPTIMIZE FOR n ROWS clause will not limit the size of the answer set).

3.5.3 Optimization Class

When an SQL statement is compiled, the optimizer will determine the access plan for that query. To achieve a balance between the improvement in execution performance of the query and the amount of resources used by the optimizer to determine the best access plan, the concept of optimization class is used.

The optimization class limits the amount of resources used by the optimizer. To set a specific optimization class, you should do the following:

1. For static SQL statements:

   When precompiling or binding, use the QUERYOPT option of these commands.
2. For dynamic SQL statements:

   Use the SET CURRENT QUERY OPTIMIZATION SQL command.

There are several optimization classes. By default, the optimizer will use class 5. The available classes are:

- Class 0

  The optimizer uses minimum resources to optimize the query. This is suitable for simple dynamic SQL queries accessing indexed tables. List prefetch is not even considered as an access method; only basic query rewrite rules are considered.

- Class 1

  List prefetch is disabled as an access method, and only a subset of query rewrite rules are applied.
- Class 3

  Most query rewrite rules are applied; list prefetch is considered as a possible access method.
- Class 5

  All query rewrite rules are applied (except computationally intensive rules). For dynamic SQL statements, optimization is reduced, and all of the required resources are available. This optimization class is suitable for mixed environments (transactions + complex queries).
- Class 7

  Same as class 5, without any reduction in optimization for dynamic SQL statements.
- Class 9

  Maximum level of optimization. Appropriate for very long and very complex queries using large tables.

Notice that the size of the statement heap can affect the amount of optimization that will be performed for complex SQL statements. If `stmtheap` is not large enough, the optimization class may be 'downgraded' by the optimizer. If this occurs, an SQL warning is received.

3.5.4 Rebinding Applications

To assure that the best access plan is used by the SQL statements to access data, packages should be rebound after changing the settings of the following database configuration parameters or executing runstats:

- `buffpage`
- `sortheap`
- `locklist`
- `maxlocks`
- `seqdetect`
- `avg_appls`

DB2 common server provides a utility, `db2rbind`, that rebinds all packages of a database. It requires SYSADM or DBADM authority. The `bind` or the `rebind` commands could also be used.

Chapter 4. Database Monitoring

This chapter discusses the different methods available to monitor the database activity, using tools such as the snapshot monitor, the event monitor and the performance monitor. Based on the information provided by these tools, you can make informed decisions about what actions need to be taken to tune the database environment.

4.1 Overview

Understanding your database and applications helps you tune your database and improve its performance. To do this, you may need to monitor your database to gather performance and statistical information about the operations and tasks that are executing.

There are three tools available with DB2 that can be used for monitoring your database. These are the snapshot monitor, the event monitor and the performance monitor.

The DB2 snapshot monitor can capture performance information at periodic points of time. The DB2 event monitor is designed to provide a summary of activity at the completion of events such as statement execution, transaction completion or when an application disconnects. Finally, the performance monitor can provide you with real-time performance data for the objects you request.

In the following sections, we discuss the use of each of these tools, and how to interpret the output.

4.1.1 Before You Start

The following items are several key points which may help you to understand the importance of database monitoring. Using database monitoring can help you to:

- Determine the source and cause of problems

  You can monitor different levels of databases such as database locking or tablespaces activity. By analyzing the output data, it may be easier to find possible problems.

- Tune configuration parameters

  Based on the monitored data, you may correct any unsuitable configuration values.

- Improve database and application performance

  After figuring out any problems in your databases or applications, it may be possible to modify them to produce better overall performance.

- Better understand user/application activity

  Database monitoring provides you information about applications. You can understand what the database manager is doing and how the database manager deals with the applications. This will help you to comprehend the activities occurring in databases.

For better use of database monitoring, you will need to perform some preliminary steps.

Figure 16 outlines the steps that you should go through for monitoring and tuning your database:

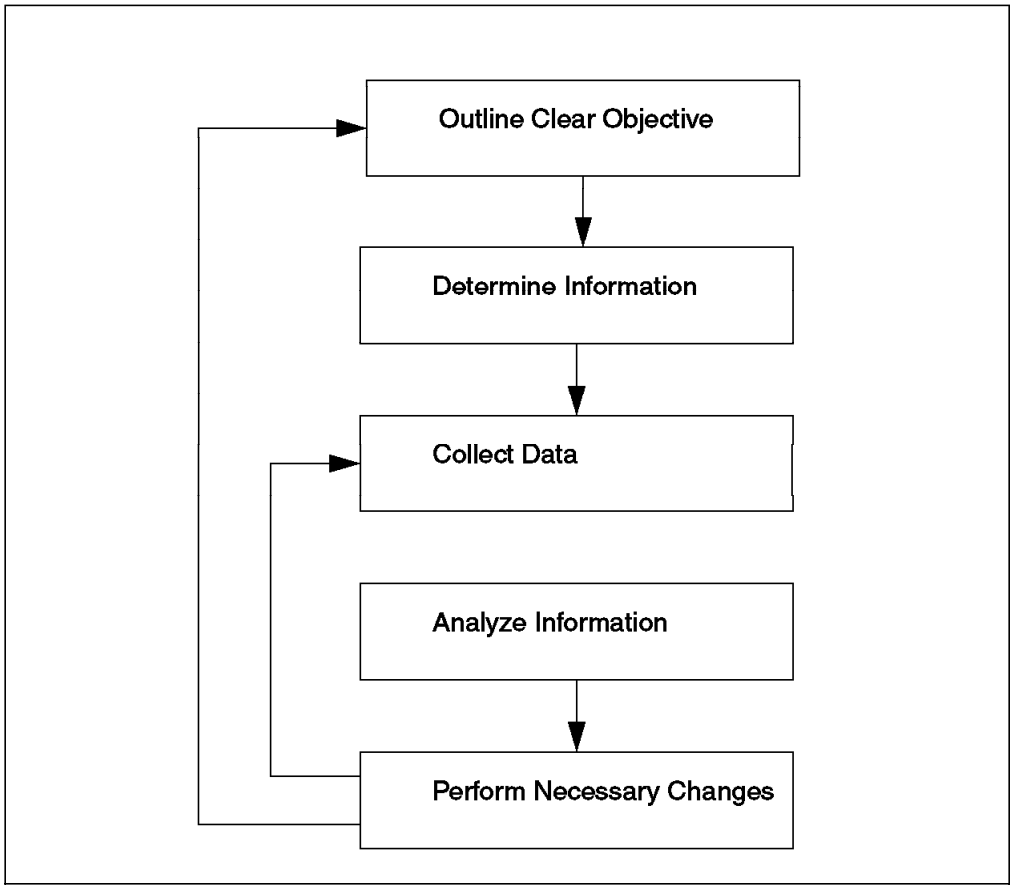


Figure 16. Database Monitoring Procedures

1. Outline Clear Objective

   If you feel that there are some problems with your database performance, you will need to make your objective clear. Your objective could be to improve the performance of a specific application or to modify the database configuration file values after some change to your database environment.

2. Determine the Type of Data Required

   Once you know your objective, you will need to decide on the type of information you wish to monitor. The type of information you require may dictate which if the different tools you will use to monitor your database. For example, choosing the snapshot monitor when you want to analyze the buffer pool usage, the amount of lock escalations or gather overall I/O information for the entire database. Likewise, you would choose the event monitor if you wanted to analyze the resources used for a query or record all deadlocks.

3. Assure Valid Data Collection

   When collected data is to be analyzed, it is important that it reflects true database activity. The analysis can only be as accurate as the data collected. You may decide upon collecting data every hour, or at random

time intervals. You need to be sure that the data is collected during times that possible problems are likely to occur.

4. Analyze Information

   After gathering the data from the available tools, you will need to analyze the information. You may use the data you collect now as a base that can then be compared with the future data. It is also possible to graph the collected data to show overall trends in area's such as lock usage or wait time. Some times to get the complete picture of the data, you need to check both snapshot data and event monitor data.

5. Perform Necessary Changes

   Based on you analysis of the information obtained and your understanding, you may decide to modify the database environment. This may involve a hardware change, network modifications or changes to the database or database manager parameters.

It is important to remember that after making some changes to the database, you should monitor the effect closely. Sometimes, a small change can have a large impact on your system, and this effect could be positive or negative.

4.2 Snapshot Monitor

The snapshot monitor provides you with information on database activity at a specific point in time. The snapshot monitor will collect information on objects like database heaps, buffers, locks, transactions and table activity. The amount of information that is collected by the snapshot monitor is determined by a number of switches that can be turned on or off at the database manager level.

4.2.1 Configuring the Snapshot Monitor

To use the snapshot monitor, you must have SYSMAINT, SYSCTRL or SYSADM authority for the instance you wish to monitor. If you have this authority, then you will be able to take a snapshot using the Command Line Processor (CLP), or through the Database Director utility. It is also possible to take a snapshot using the database system monitor APIs, and thus develop applications that collect the information you require.

The snapshot monitor provides comprehensive and flexible data collection; you can monitor over 150 performance variables using different data objects and levels. The snapshot monitor breaks the information that can be collected into a number of groups, known as monitor groups, which are set at the database manager level. The snapshot monitor groups include:

- Basic
- Sort
- Lock
- Table
- Buffer Pool
- Unit of Work
- SQL Statement

Within these groups, it is possible to define threshold values for each of the variables being monitored. When these threshold values are reached, an action can be triggered. This action may be a program or script that will be executed by the monitor program. This is discussed in 4.4.2.1, “Monitoring Snapshot Data” on page 72.

4.2.1.1 Basic Monitor Group

The basic monitor group information is collected by default. The information collected by this group includes the following elements:

- General information

  The general information includes the configuration name of the monitored node, a time-stamp for when the monitoring information was taken, the database name and path, and the type of snapshot monitor data.

- Database connections

  The status of local and remote database connections are monitored, along with additional information such as the total number of connections and the maximum number of concurrent connections. This information will help you to determine the level of concurrent processing that occurs in the database manager.

- Locks and deadlocks

  This element includes the total lock list memory, lock escalations, current applications waiting on locks, the lock status and number of lock time-outs. It can help you to analyze any resource contention problems or application concurrency problems.

- SQL statement activity

  SQL statement activity includes information on both static and dynamic SQL statements. Information such as commit statements attempted is recorded along with information on update, insert or delete SQL statements that are executed and rollbacks that have been attempted.

- Sort work

  Includes total sort heaps allocated, active sorts, and the piped sorts requested and accepted. It can help you to determine any sort heap problems.

- SQL cursors

  Includes information about open local and remote cursors. You may calculate the percentage of local cursors that are blocking cursors. If the percentage is low, you may improve performance by improving the row blocking in the application.

- Communication activity

  Includes information about current communication heap size and maximum communication heap size. Use this element to understand the memory requirements for an application.

- Table activity

  Table activity information includes rows that are inserted, updated or deleted. This element can help you gain insight into activity within the database manager.

- Database activity

This element includes information about binds and precompiles attempted, time-stamp for the last backup and the maximum database heap allocated. Using this element can help you to evaluate the related database configuration parameters.

- Agents and applications

  Includes information about the agents registered, maximum number of agents registered or waiting, application status, ID and name, client process ID, communication protocol, and the number of idle agents.

- CPU usage

  Information about the CPU includes user CPU time used by agent and system CPU time used by agent. This information will help you to identify applications that could possibly benefit from additional tuning.

- Logging

  Includes the maximum total log space used, maximum secondary log space used and number of log pages read/written. This information will help you to evaluate your configuration settings and determine appropriate settings for parameters such as `logfilsz`, `logprimary`, `logsecond`, `logretain`.

- Caching

  Includes information about catalog cache inserts, lookups and overflows. You can calculate the package cache hit ratio, and it will help you to see if the package cache is being used effectively.

For more information about the individual data elements, refer to *DB2 Database System Monitor Guide and Reference - for common servers* (S20H-4871).

4.2.1.2 Additional Monitor Groups

If you wish to collect information in addition to that supplied by the basic monitor group, then you need to turn on one or more of the following monitor groups. Detailed information about these groups can be found in the manual *DB2 Database System Monitor Guide and Reference - for common servers* (S20H-4871).

Buffer Pool Monitor Group: The buffer pool monitor group includes information about the number of reads, writes and time taken for data and index operations. These data elements help you to measure bufferpool activities, such as the hit ratio of the bufferpool and the prefetcher page number.

Lock Monitor Group: This group includes information on the number of lock waits, types of locks, IDs of applications holding the locks and lock wait start time-stamp. These data elements can help you to analyze the lock situation and lock escalations that occur.

Sort Monitor Group: The sort group includes post threshold sorts, total sort time, overflows and total sorts performed. It can help you to check if the sort heap value is suitable.

Statement Monitor Group: The statement monitor group includes the statement type, start time-stamp, end time-stamp, user CPU time used by statement and system CPU time used by statement. This will help you determine if your statements need to be further optimized.

Table Monitor Group: This includes table types, rows read from the table since connect and rows written to the table since connect. This will help you to measure the activities of tables, such as if the table needs to be reorganized using the reorg command.

Unit of Work Monitor Group: This includes total time the units of work waited on locks, the start timestamp and end timestamp, completion status and log space used. This information will help you to analyze the system contention problems, to determine the reason a unit of work ended (due to deadlock or abnormal termination).

By default, all these switches are off. The associated information will be collected when the specific group switch is turned on; otherwise, only information for the basic monitor group will be collected. When you turn a group off, all the different data elements related to that group are reset. It is also possible to reset the counters related to a monitor group, without turning the group off. This is covered in 4.2.1.3, “Monitor Counters.”

It is possible to turn multiple switches on or off at the same time. However, if you turn on or off a single monitor group, it will not affect other monitor groups’ data values.

4.2.1.3 Monitor Counters

Counters are used by the snapshot monitor to keep track of the different activities/elements that are to be measured. These counters increase in value over time, and because of this, there may be times when you will need to reset the values.

Understanding when counting starts may help you select a suitable time to take a snapshot. Counting can start at different times in response to the following:

1. Application connection to the database
    - At the application level, counting starts when the application connects
    - At the database level, counting starts when the first application connects
    - At the table level, counting starts upon the first table access
    - At the tablespace level, counting starts upon the first tablespace access
2. Last counter reset
3. Responsible monitor group turned on

As an example, if you turn on a table switch for a table, no snapshot data for that table will be collected until after the first access to that table.

Resetting Counters: There may be times when you need to reset the monitor’s counters. For example, you may want to compare values from the same monitor group over two different intervals in time. There are three ways to reset counters:

1. Using CLP command

   ```
   RESET MONITOR {ALL | FOR DATABASE database-alias}
   ```

2. Using APIs from an application

   It is possible to use the API call, sqlmrset, to reset system monitor data areas.

3. Using Performance Monitor

   You can reset all or selected counter data elements. For detailed information, refer to the Getting Started option found in the on-line help for the Database Director.

4.2.2 Snapshot Monitor Commands

To execute the snapshot monitor commands, you will require SYSADM, SYSCTRL or SYSMAINT authority.

The first task you will need to perform when preparing to take a database snapshot is to check the current monitor switches that are associated with the different monitor groups. To check the current monitor switch settings, you can use the DB2 command `get monitor switches`, as shown in Figure 17.

```
$db2 get monitor switches

            Monitor Recording Switches

Buffer Pool Activity Information (BUFFERPOOL) = ON 07-02-1996 12:01:37.89
Lock Information                         (LOCK) = OFF
Sorting Information                      (SORT) = OFF
SQL Statement Information           (STATEMENT) = OFF
Table Activity Information              (TABLE) = OFF
Unit of Work Information                  (UOW) = OFF
```

Figure 17. Getting Current Monitor Settings

To turn on or off the monitor switches, you use the command:

```
UPDATE MONITOR SWITCHES USING {switch-name {ON | OFF}...}
 switch-name:
 BUFFERPOOL, LOCK, SORT, STATEMENT, TABLE, UOW
```

If you wish to reset the monitor elements back to zero, then the following command can be used:

```
RESET MONITOR {ALL | FOR DATABASE database-alias}
```

This command can specify a particular database or all databases. It is not possible to specify the individual monitor group using the reset command. To reset counters for a specific monitor group, you must turn that monitor group switch off and then back on.

You should remember that if a monitor group is not turned on, then the information supplied for that group by the snapshot tool will not be complete, as only the basic monitor information is collected by default.

Once you have turned the appropriate monitor group on, you are ready to take a snapshot while database activity is occurring. The command to take a snapshot is:

```
GET SNAPSHOT FOR {DATABASE MANAGER | ALL DATABASES | ALL APPLICATIONS |
APPLICATION {APPLID appl-id | AGENTID agentid} | {ALL | DATABASE |
APPLICATIONS | TABLES | TABLESPACES | LOCKS} ON database-alias}
```

It is possible to obtain a snapshot from a remote instance by attaching to that instance before taking the snapshot.

It is also possible to perform the above operations using the Database Director. From the Database Director, you will need to configure the database instance or node to enable the monitor groups whose information you wish to collect, as shown in Figure 18 on page 52.

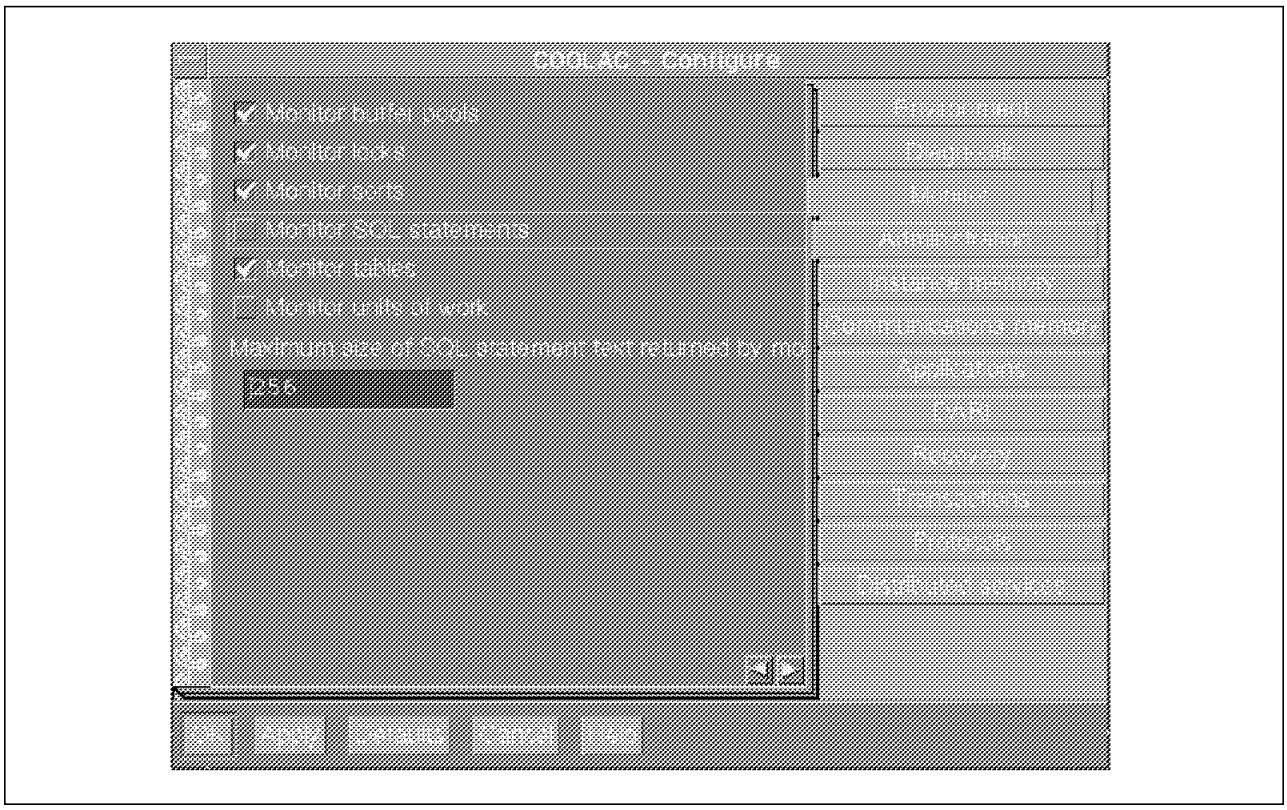


Figure 18. Database Director - Enabling Monitor Groups

4.4, “Performance Monitor” on page 71, discusses how snapshot monitoring can be done outside the Command Line Processor.

4.2.3 Taking a Snapshot

Once you have set the appropriate monitor switches, you are ready to take snapshots of you database environment. There are six levels of snapshots that you may perform. These levels are:

- Database manager level, for a snapshot of the database manager instance.
- Database level, for collecting information on all or selected databases.
- Application level, for all or selected applications.
- Table level, for tables accessed while the table switch/group is on.
- Lock level, for locks held by connected applications.
- Tablespace level, for tablespaces accessed while the bufferpool switch/group is on.

Once you have chosen the level at which you wish to take the snapshot, there are three ways that this may be done.

The first method for taking a snapshot is by using the Command Line Processor and the command `get snapshot`. The second method is using the Database

Director/performance monitor, which is covered in 4.4, “Performance Monitor” on page 71. The final method is to use the available monitor APIs. Figure 19 on page 53 compares the available APIs to the equivalent Command Line Processor commands that perform the monitor functions.

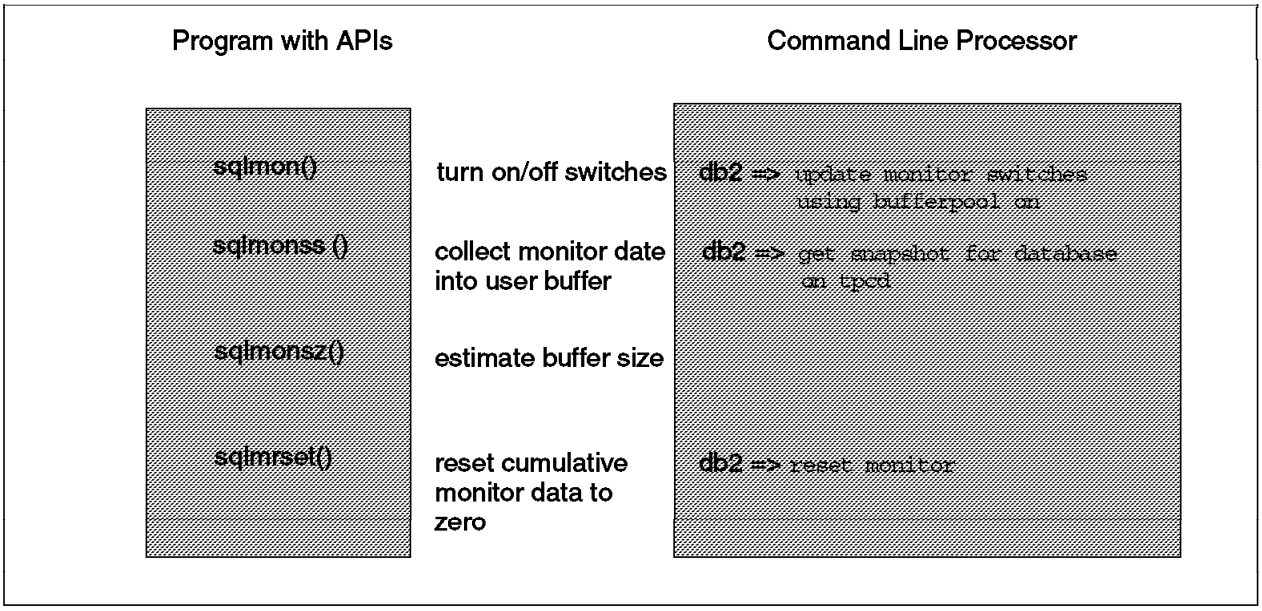


Figure 19. Monitor APIs and CLP Commands

When programming with APIs, you need to call some routines such as sqlmon() to turn on/off switches. It is important to note that, when using the API to take a snapshot, you need to manually determine the buffer sizes that will be used. This is done using the sqlmonsz() API. When using either of the other two methods to take a snapshot, the buffersize is automatically estimated. For information about the APIs you should refer to the DB2 *Database System Monitor Guide and Reference - for common servers*.

4.2.4 Interpreting Snapshot Output

The output from the snapshot monitor provides you with detailed information about the database activity. From this data you can make an informed decision about any actions that are required to improve the performance of your database environment.

This section shows you how to interpret the information returned to you by the snapshot monitor, and which database parameters should be modified, based on this data, to improve the performance.

As mentioned previously, there are different levels of detail available through the snapshot monitor. The following examples look at these different levels. Some of the elements in the snapshot may not have any associated values; this may be because the monitor group has been turned off or because there has been no activity on that element since monitoring was enabled.

GET SNAPSHOT FOR DATABASE MANAGER: From the database manager snapshot, shown in Figure 20 on page 54, we can see that the instance is a server instance supporting local and remote clients. We can see that there are

two remote connections to the database and one of the connections is executing in the database manager. This means that only one of these connections is executing a unit of work within the instance being monitored; in this example, the instance db2.

```
 Database Manager Snapshot

Node type                                    = Database Server with local
and remote clients
Instance name                                 = db2
Database manager status                       = Active

Sort heap allocated                           = 0
Post threshold sorts                          = 0
Piped sorts requested                         = 8
Piped sorts accepted                          = 8

Start Database Manager timestamp              = 09-23-1996 17:43:15.640390
Last reset timestamp                          =
Snapshot timestamp                            = 09-24-1996 15:08:33.915171

Remote connections to db manager              = 2
Remote connections executing in db manager    = 1
Local connections                             = 0
Local connections executing in db manager     = 0
Active local databases                        = 1

High water mark for agents registered         = 7
High water mark for agents waiting for a token = 0
Agents registered                             = 7
Agents waiting for a token                    = 0
Idle agents                                   = 3

Committed private Memory (Bytes)              = 212992
```

Figure 20. Snapshot for Database Manager

The snapshot also indicates that all pipe sort requests have been accepted, and that no sort heap is allocated. We also can infer from the post threshold sort value that no sort requests were received while the sort heap threshold was reached. If this were not the case, then you might need to tune the database manager sort heap size parameter, sheapthres, or the database configuration parameter, sortheap.

GET SNAPSHOT FOR DATABASE: The database snapshot provides an overall look at the activity of the individual database you wish to monitor. The first part of the database monitor output, as shown in Figure 21 on page 55, includes the name and location of the database.

We can also see the locking status of the database. At the time the snapshot was taken, there were 18 locks being held, and no application was waiting for a lock. There have been no lock escalations, and so we can be fairly sure that the locking configuration parameters are set high enough to handle the database activity. If the number of lock escalations had been high, then you might have considered increasing the maxlocks database configuration parameter.

```
Database Snapshot

Database name                               = TPCD
Database path                               = /usr/data/db2/SQL00002/
Input database alias                        = TPCD
Database status                             = Active

Locks held currently                        = 18
Lock waits                                  = 0
Time database waited on locks (ms)          = 0
Lock list memory in use (Bytes)             = 2268
Deadlocks detected                          = 0
Lock escalations                            = 0
Exclusive lock escalations                  = 0
Current applications waiting on locks       = 0
Lock Timeouts                               = 0

Total sort heap allocated                   = 0
Total sorts                                 = 1
Total sort time (ms)                        = 189
Sort overflows                              = 0
Active sorts                                = 0
```

Figure 21. Snapshot for Database (Part 1 - Database, Sorts and Locks)

From the database snapshot, we can also see the total number of sorts that occurred and the number of overflows. An overflow will occur when there is insufficient sort heap space to perform the sort, and disk space was used for temporary storage. If this occurs, then you should look at increasing the sort heap size at the database and/or instance level.

Figure 22 on page 56 is a continuation of the output from the database snapshot, and this includes detailed information about the buffer pools and database I/O that has occurred. The information about buffer pool I/O and direct I/O can help you to tune the buffer size and should increase overall performance of read and writes for transactions. 3.1.1, “Database Buffer Pool” on page 13, provides further information on how these values can be used to estimate buffer pool hit ratios.

The final section of output from the database snapshot is shown in Figure 23 on page 57. This includes information about the types of statements and applications that are being executed against the database and the operations that are being performed. This includes information such as the success of the transactions; wether they were committed or rolled back. You can also see if statements were dynamic or static and how many rows were read, inserted, updated or deleted.

This type of information can help you make decisions such as whether commits should be grouped in the database environment.

```
Buffer pool data logical reads              = 1934
Buffer pool data physical reads             = 131
Asynchronous pool data page reads           = 108
Buffer pool data writes                     = 0
Asynchronous pool data page writes          = 0
Buffer pool index logical reads             = 119
Buffer pool index physical reads            = 28
Buffer pool index writes                    = 0
Asynchronous pool index page writes         = 0
Total buffer pool read time (ms)            = 1488
Total buffer pool write time (ms)           = 0
Total elapsed asynchronous read time        = 333
Total elapsed asynchronous write time       = 0
Asynchronous read requests                  = 6
LSN Gap cleaner triggers                    = 0
Dirty page steal cleaner triggers           = 0
Dirty page threshold cleaner triggers       = 0

Direct reads                                = 74
Direct writes                               = 0
Direct read requests                        = 10
Direct write requests                       = 0
Direct reads elapsed time (ms)              = 256
Direct write elapsed time (ms)              = 0

Database files closed                       = 0
```

Figure 22. Snapshot for Database (Part 2 - Buffers and I/O)

If you find that the database is performing a large amount of selects and few inserts, updates or deletes, then you may decide to stop archival logging on that database and add more table indexes to see if you are able to increase the response for queries.

GET SNAPSHOT FOR APPLICATIONS: If you find that a particular application is performing poorly, then an application snapshot can provide you with information about what is happening.

Figure 24 on page 58 show the output from a snapshot for a remote connection of an application running under the AIX operating system. From the snapshot, we can determine the database being accessed, the application process ID and both the authorization and execution ID being used.

From this snapshot, we can also see locking information about the application. This shows the number of locks that the application is holding, plus the number of times the application was required to wait for a lock and any deadlock situations that have been detected. The database snapshot also lists deadlock information, which can be used in conjunction with this information to help narrow down any possible deadlock problems.

```
Commit statements attempted                = 6
Rollback statements attempted              = 2
Dynamic statements attempted               = 1963
Static statements attempted                = 8
Failed statement operations                = 3
Select SQL statements executed             = 3
Update/Insert/Delete statements executed   = 0

DDL statements executed                    = 0
Internal automatic rebinds                 = 0
Internal rows deleted                      = 0
Internal rows inserted                     = 0
Internal rows updated                      = 0
Internal commits                           = 2
Internal rollbacks                         = 0
Internal rollbacks due to deadlock         = 0

Rows deleted                               = 0
Rows inserted                              = 0
Rows updated                               = 0
Rows selected                              = 1946

Binds/precompiles attempted                = 0

First database connect timestamp           = 09-24-1996 15:05:23.616990
Last reset timestamp                       =
Last backup timestamp                      = 09-24-1996 14:05:13.041583
Snapshot timestamp                         = 09-24-1996 15:08:33.996282

High water mark for connections            = 2
High water mark for database heap          = 413687
Application connects                       = 2
Applications connected currently           = 2
Appls. executing in db manager currently   = 0

Maximum secondary log space used (Bytes)   = 0
Maximum total log space used (Bytes)       = 1920
Secondary logs allocated currently         = 0
Log pages read                             = 0
Log pages written                          = 4

Package cache lookups                      = 13
Package cache inserts                      = 3

Catalog cache lookups                      = 8
Catalog cache inserts                      = 5
Catalog cache overflows                    = 0
Catalog cache heap full                    = 0
```

Figure 23. Snapshot for Database (Part 3 - Statements, packages and catalogs)

As with the database snapshot, the application snapshot also provides details about the buffer pool and direct I/O requests. If you are receiving a poor buffer pool hit ratio at the database level, then you may be able to narrow this problem down to the individual application. You may find that by tuning some individual applications, the overall database hit ratio will be improved.

```
Application Snapshot

Agent ID                                    = 17800
Application status                          = UOW Waiting
Status change time                          = 09-24-1996 15:08:34.293534
ID of code page used by application         = 819
Country code of database                    = 1
DUOW correlation token                      = *TCPIP.8123DF3F.960924200523
Application name                            = db2bp_32
Application ID                              = *TCPIP.8123DF3F.960924200523
Sequence number                             = 0001
Authorization ID                            = RUSCONI
Execution ID                                = rusconi
Configuration NNAME of client               = gundagai
Client database manager product ID          = SQL02011
Process ID of client application            = 3576
Platform of client application              = AIX
Communication protocol of client            = TCP/IP
Database name                               = TPCD
Database path                               = /usr/data/db2/SQL00002/
Client database alias                       = tpcd
Input database alias                        = TPCD

Locks held by application                   = 18
Lock waits since connect                    = 0
Time application waited on locks (ms)       = 0
Deadlocks detected                          = 0
Lock escalations                            = 0
Exclusive lock escalations                  = 0
Number of Lock Timeouts since connected     = 0

Total time UOW waited on locks (ms)         = 0
ID of Agent holding lock                    =
Application ID holding lock                 =
Sequence number holding lock                =
Name of Tablespace holding lock             =
Schema of Table holding lock                =
Name of Table holding lock                  =
Lock mode                                   =
Lock object type                            =
Lock object name                            =
Lock wait start timestamp                   =
```

Figure 24. Snapshot for Applications (Part 1 - Application and locks)

Figure 26 on page 60 shows the statement and UOW information that is returned by the application snapshot. This information can be used to help track what an application is doing in terms of database access, and it may in turn help you to determine where application or database improvements may be made.

```
Total sorts                                 = 1
Total sort time (ms)                        = 189
Total sort overflows                        = 0

Buffer pool data logical reads              = 2006
Buffer pool data physical reads             = 21
Buffer pool data writes                     = 0
Buffer pool index logical reads             = 83
Buffer pool index physical reads            = 28
Buffer pool index writes                    = 0
Total buffer pool read time (ms)            = 900
Total buffer pool write time (ms)           = 0

Direct reads                                = 38
Direct writes                               = 0
Direct read requests                        = 5
Direct write requests                       = 0
Direct reads elapsed time (ms)              = 201
Direct write elapsed time (ms)              = 0

Commit statements                           = 1
Rollback statements                         = 0
Dynamic SQL statements attempted            = 2049
Static SQL statements attempted             = 1
Failed statement operations                 = 1
Select SQL statements executed              = 3
Update/Insert/Delete statements executed    = 0

DDL statements executed                     = 0
Internal automatic rebinds                  = 0
Internal rows deleted                       = 0
Internal rows inserted                      = 0
Internal rows updated                       = 0
Internal commits                            = 1
Internal rollbacks                          = 0
Internal rollbacks due to deadlock          = 0

Rows deleted                                = 0
Rows inserted                               = 0
Rows updated                                = 0
Rows selected                               = 2038
Rows read                                   = 2052
Rows written                                = 0

Binds/precompiles attempted                 = 0
```

Figure 25. Snapshot for Applications (Part 2 - Buffers and I/O)

```
UOW log space used (Bytes)                  = 0
Previous UOW completion timestamp           = 09-24-1996 15:05:25.123601
UOW start timestamp                         = 09-24-1996 15:06:09.461284
UOW stop timestamp                          =
UOW completion status                       =

Open remote cursors                         = 1
Open remote cursors with blocking           = 1
Rejected Block Remote Cursor requests       = 0
Accepted Block Remote Cursor requests       = 3

Open local cursors                          = 0
Open local cursors with blocking            = 0

Current communication heap size (Bytes)     = 0
Maximum communication heap size (Bytes)     = 0

Connection request start timestamp          = 09-24-1996 15:05:23.616990
Connect request completion timestamp        = 09-24-1996 15:05:25.104819
Last reset timestamp                        =
Snapshot timestamp                          = 09-24-1996 15:08:34.391125

Statement type                              = Dynamic SQL Statement
Statement operation                         = Fetch
Section number                              =
Application creator                         =
Package Name                                =
Cursor name                                 = SQLCUR201
Statement sorts                             = 0
Statement operation start timestamp         = 09-24-1996 15:08:34.292796
Statement operation stop timestamp          = 09-24-1996 15:08:34.293570

Total User CPU Time used by statement (s)   = 0.000000
Total System CPU Time used by statement (s)= 0.000000
Total User CPU Time used by agent (s)       = 2.830000
Total System CPU Time used by agent (s)     = 0.530000

Dynamic SQL statement text                  =
select * from tpcd.customer

Package cache lookups                       = 7
Package cache inserts                       = 3

Catalog cache overflows                     = 0
Catalog cache heap full                     = 0
```

Figure 26. Snapshot for Applications (Part 3 - UOW and statements)

GET SNAPSHOT FOR TABLES: Data placement may be a critical factor for performance. If your system is spread over multiple disks, then it would be beneficial to spread the I/O requests as evenly as possible across the disks. By looking at the most-frequently accessed tables, you may be able to reorganize these tables so they are spread out or on faster disks.

The output from the table snapshot, as shown in Figure 27 on page 61, not only shows the number of rows read and written for each of the accessed tables but also includes the number of overflows.

An overflow will usually occur when a VARCHAR column has been updated and no longer fits into the original space allocated for it. If the number of overflows becomes high, then you should consider reorganizing the table with the reorg utility.

```
 Table Snapshot

First database connect timestamp     = 09-24-1996 15:05:23.616990

Last reset timestamp                 =
Snapshot timestamp                   = 09-24-1996 15:08:35.046271
Database name                        = TPCD
Database path                        = /usr/data/db2/SQL00002/
Input database alias                 = TPCD
Number of accessed tables            = 7

Table Schema Table Name     Table Type Rows     Rows Overflows
                                       Written  Read
------------ -------------- ---------- -------- ---- ----------
TPCD         CUSTOMER       User       0        1962          0
SYSIBM       SYSTABAUTH     Catalog    4           3          0
SYSIBM       SYSTABLES      Catalog    0          85          0
SYSIBM       SYSTABLESPACES Catalog    0           6          0
SYSIBM       SYSPLANAUTH    Catalog    0           1          0
SYSIBM       SYSPLAN        Catalog    0           2          0
SYSIBM       SYSDBAUTH      Catalog    0           5          0
```

Figure 27. Snapshot for Tables

By reorganizing a fragmented table, you may improve the access time and minimize the number of actual I/O operations.

GET SNAPSHOT FOR TABLESPACES: If you require information about I/O and buffer usage at a finer level than supplied by the database snapshot, you are able to take a snapshot at the tablespace level, as shown in Figure 28 on page 62.

The tablespace snapshot includes information about the buffer pool logical and physical reads for both data and indexes.

By collecting this information and applying it to the formula displayed in Chapter 3, “DB2 Performance Considerations” on page 11, you can maximize the throughput for the individual tablespaces.

```
 Tablespace Snapshot

First database connect timestamp           = 09-24-1996 15:05:23.616990
Last reset timestamp                       =
Snapshot timestamp                         = 09-24-1996 15:08:35.146641
Database name                              = TPCD
Database path                              = /usr/data/db2/SQL00002/
Input database alias                       = TPCD
Number of accessed tablespaces             = 3

Tablespace name                            = SYSCATSPACE

  Buffer pool data logical reads           = 61
  Buffer pool data physical reads          = 19
  Asynchronous pool data page reads        = 0
  Buffer pool data writes                  = 0
  Asynchronous pool data page writes       = 0
  Buffer pool index logical reads          = 119
  Buffer pool index physical reads         = 28
  Buffer pool index writes                 = 0
  Asynchronous pool index page writes      = 0
  Total buffer pool read time (ms)         = 1095
  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                             = 74
  Direct writes                            = 0
  Direct read requests                     = 10
  Direct write requests                    = 0
  Direct reads elapsed time (ms)           = 256
  Direct write elapsed time (ms)           = 0

  Number of files closed                   = 0
```

Figure 28. Snapshot for Tablespaces

GET SNAPSHOT FOR LOCKS: Poor performance can often be due to locking problems and concurrency. By taking a snapshot of the locks, it is possible to determine if one application is allocating too many locks, causing lock waits for other applications.

```
 Lock Snapshot

Database name                              = TPCD
Database path                              = /usr/data/db2/SQL00002/
Input database alias                       = TPCD
Locks held                                 = 18
Applications currently connected           = 2
Applications currently waiting on locks    = 0
Snapshot timestamp                         = 09-24-1996 15:08:35.324131
```

Figure 29. Snapshot for Locks (Part 1 - Locks)

Figure 29 shows the general information about the lock snapshot, and Figure 30 on page 63 lists the output for one of the connected applications. In this example, the application/agent shown is holding all the locks on the database.

```
Agent ID                                  = 17800
Application ID                            = *TCPIP.8123DF3F.960924200523
Sequence number                           = 0001
Application name                          = db2bp_32
Authorization ID                          = RUSCONI
Application status                        = UOW Waiting
Status change time                        = 09-24-1996 15:08:34.293534
ID of code page used by application       = 819
Locks held                                = 18
Agent ID holding lock                     =
Application ID holding lock               =
Sequence number holding lock              =
Tablespace Name of lock                   =
Table Schema of lock                      =
Table Name of lock                        =
Lock object type waited on                =
Lock object name                          =


Object Object
Name   Type   Tablespace Name Table Schema Table Name     Mode Status
------ ------ -------------- ------------ -------------- ---- ----------
23045  Row                   TPCD         CUSTOMER       S    Granted
7      Table                 TPCD         CUSTOMER       IS   Granted
286    Row                   SYSIBM       SYSTABAUTH     S    Granted
1286   Row                   SYSIBM       SYSTABLES      S    Granted
4      Row                   SYSIBM       SYSTABAUTH     S    Granted
13     Table                 SYSIBM       SYSTABAUTH     IS   Granted
4      Row                   SYSIBM       SYSTABLES      S    Granted
2      Table                 SYSIBM       SYSTABLES      IS   Granted
28     Table                 SYSIBM       SYSTABLESPACES S    Granted
22     Row                   SYSIBM       SYSPLANAUTH    S    Granted
0      Row                   SYSIBM       SYSPLANAUTH    S    Granted
12     Table                 SYSIBM       SYSPLANAUTH    IS   Granted
0      Row                   SYSIBM       SYSDBAUTH      S    Granted
5      Row                   SYSIBM       SYSDBAUTH      S    Granted
4      Row                   SYSIBM       SYSDBAUTH      S    Granted
11     Table                 SYSIBM       SYSDBAUTH      IS   Granted
9      Row                   SYSIBM       SYSPLAN        S    Granted
7      Table                 SYSIBM       SYSPLAN        IS   Granted
```

Figure 30. Snapshot for Locks (Part 2 - Application/Agent)

4.3 Event Monitor

While the snapshot monitor allows you to look at the state of the database environment when you take the snapshot, the event monitor will allow you to collect database information when a particular event or transition occurs within the database. For example, you do not usually know when a deadlock situation is going to occur or at what point in time a transaction will complete. If you wish to monitor this type of activity in the database, then you need to define an event monitor.

The different events that can be monitored include:

- Deadlocks
- Connections
- Transactions
- Statements
- Database
- Tablespaces
- Tables

The event monitor can provide you with information about your database when the event you wish to monitor occurs.

Snapshots collect information at both the database manager and the database levels, and the event monitor collects data for a single database. You may have a multiple event monitors defined at any time, and they may be turned on or off independently of each other.

To perform the event monitoring, you must have SYSMAINT, SYSCTRL or SYSADM authority for the database you wish to monitor. Event monitor data can be written to either a file or a named pipe.

Event monitors can be created by using the Command Line Processor or though an application using embedded SQL statements.

4.3.1 Using Event Monitoring

As with the snapshot monitor, you should have an idea of the types of information that you wish to collect. As with any monitoring tool, there is some impact on the system, and unnecessary monitoring means unnecessary overhead for your database environment.

As mentioned previously, the event monitors can be defined for different types of events. You should define an event monitor to collect the information that you require. Your choices include:

- Deadlock Monitor

  A deadlock monitor records information about the different resources and applications that are involved in the deadlock situation. This information includes the time the deadlock was connected, the application IDs involved in the deadlock, the locked object details plus table and tablespace details.

- Connections Monitor

  This monitor collects information on connected applications. The information collected includes the application's name, execution ID and the time the application connected. Details on the CPU time, sort time and any lock wait time is recorded.

- Transaction Monitor

  This specifies that the event monitor should record each completed transaction. The type of information recorded against a transaction includes the application details, start and stop times, and the completion status.

- Statement Monitor

Transactions may include multiple statements, which are also recorded through a statement monitor. This will record details such as the application ID, type of SQL statement and operation, the package name, creator and section number associated with the statement. Other details include the start and stop times, the elapsed time and total CPU time as well as the actual text of the SQL statement.

- Database Monitor

  This specifies that the event monitor should record a database event when the first application connects and when the last application disconnects from the database.

- Tablespace Monitor

  This directs the event monitor to record a tablespace event for each active tablespace when the last application disconnects from the database.

- Table Monitor

  This directs the event monitor to record a table event for each active table when the last application disconnects from the database.

Once you have selected the event monitor or monitors that you wish to run, you need to decide where the information is going to be written. This could be to a named pipe or to a directory.

4.3.1.1, “Event Monitor Commands,” lists the commands to create and activate an event monitor. You need to create the event monitor and set its state to active before any monitoring will begin. You are able to have any number of event monitors defined, but a maximum of 32 event monitors can be active at a given time.

4.3.1.1 Event Monitor Commands

An event monitor can be created either through the Command Line Processor or through an application using embedded SQL. You will require SYSADM or DBADM authority to create an event.

The command used to create an event is shown in Figure 31 on page 66. For the full syntax, refer to *DB2 SQL Reference - for common servers*.

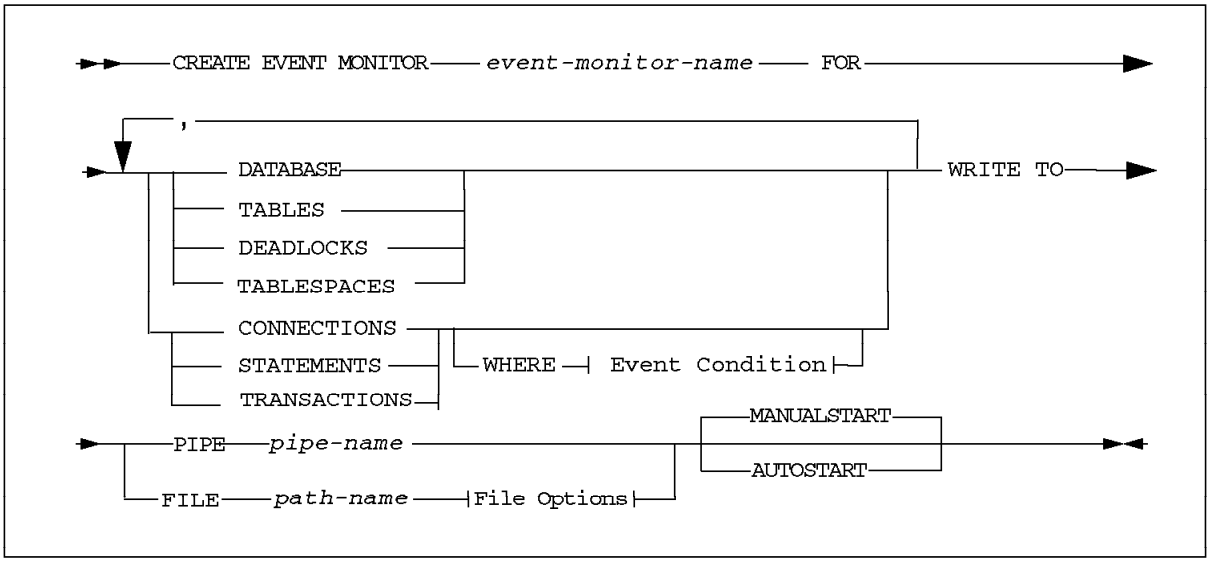


Figure 31. Create Event Monitor Command Syntax

There are many options in this command; you can select the information level and use the WHERE clause to filter the data that is recorded for connections, transactions, and SQL statements. Use the WRITE TO option to control where the data is written (using files or named pipes).

The event monitor definitions are kept in the system catalog tables, *SYSCAT.EVENTMONITORS* and *SYSCAT.EVENTS*. It is possible to check event monitor information by performing an SQL query against these tables. For example, to find the names of defined event monitors, you can use;

```
select evmonname from syscat.eventmonitors
```

If you select the AUTOSTART option for the event monitors, then they will be started each time the database is started.

The command shown in Figure 32 creates an event monitor that will monitor all the possible events for the database that it is defined in.

```
CREATE EVENT MONITOR monitor_all FOR
    database, tables, deadlocks, tablespaces,
   connections, statements, transactions
WRITE TO FILE
   /usr/data/monitor
AUTOSTART
```

Figure 32. Sample Event Monitor

These events captured with the monitor will be written into the directory, /usr/data/monitor, which must exist before monitoring is started. Once the monitor has been defined, it must be set to the active state using the command:

```
SET EVENT MONITOR event-monitor-name STATE = 1
```

This command will activate the event monitor. To determine the current state of event monitors, you need to use the event_mon_state built-in function. An example of this is shown in Figure 33 on page 67.

A state of 0 indicates the event monitor is inactive, while a state of 1 indicates and active event monitor.

```
SELECT  evmonname NAME, event_mon_state(evmonname) STATE
FROM    syscat.eventmonitors;

NAME                STATE
------------------- -----------
CNMON1                        0
DBMON1                        0
DLMON1                        0
MON_ALL                       1
STMON1                        1
TBMON1                        1
TRMON1                        1
TSMON1                        1

  8 record(s) selected.
```

Figure 33. Display Event Monitor States

It is possible to delete an event monitor from the database if you do not intend to use it again. You should first set its state to 0, to make it inactive, then use the following command to delete its definition:

```
DROP EVENT MONITOR event-monitor-name
```

4.3.2 Analyze the Output of Event Monitor

There are two utilities provided to help you analyze the output generated by the event monitor.

The db2evmon produces formatted text output that can be browsed for the information that you require. A sample of the output generated by this utility is shown in Figure 36 on page 69.

The second method for analyzing the output from the event monitor is to use the graphical tool, db2eva. This tool displays the information collected as a series of tables which may be filtered dynamically. Figure 34 on page 70 is an example of the first window the db2eva tool displays.

You are able to expand any of the monitored time periods as connection, deadlocks, deadlocked connections, overflows, transactions or statements. To do
this, select the appropriate monitored time period and use the Selected option at the top of the window. Alternatively you may double-click any of the time periods,
and you will open the connection window automatically.

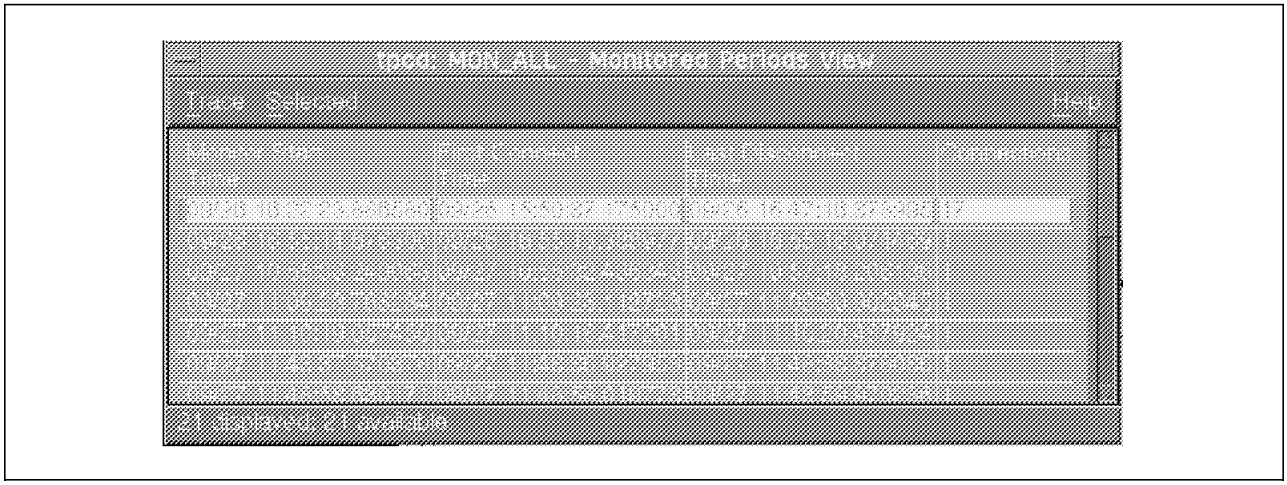


Figure 34. Event Monitor Tool, db2eva - Monitored Periods

From the connections window, it is possible to see when applications connected, whether they were local or remote, the application name and connection ID and times taken for sorting, waiting on locks and CPU usage.

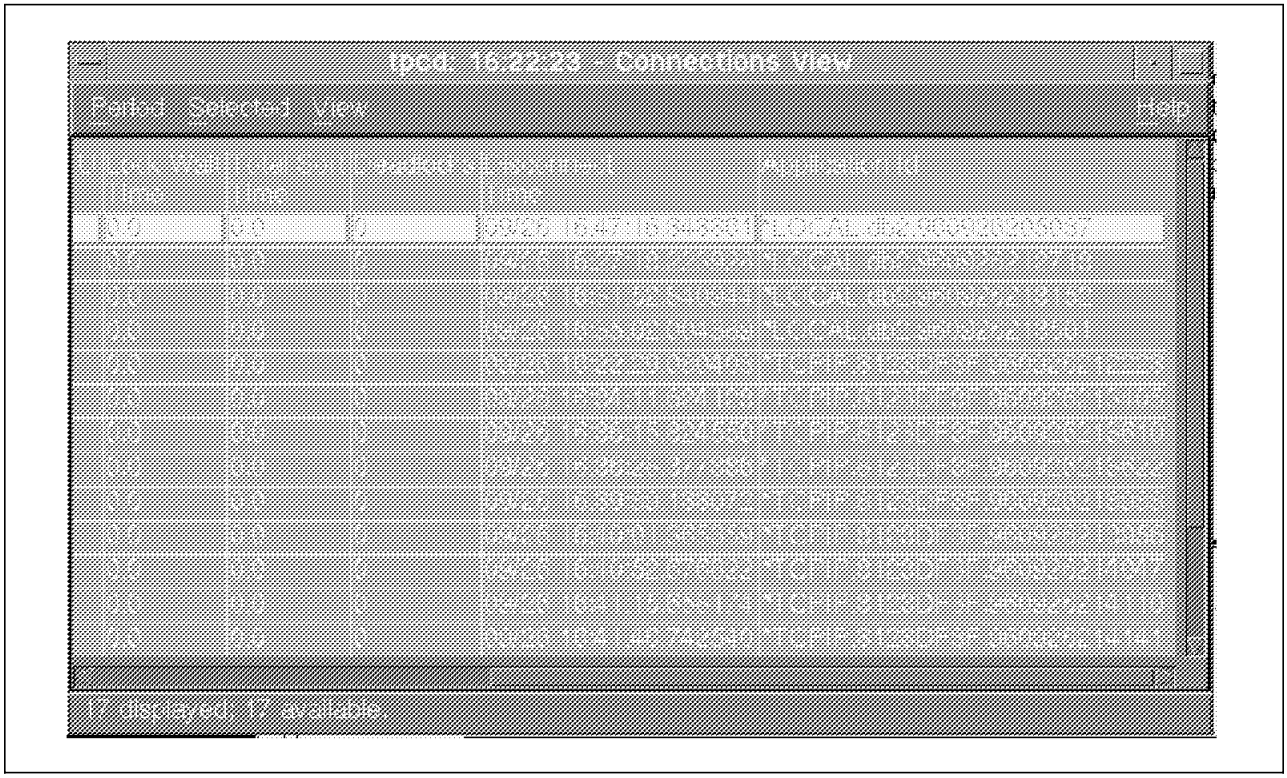


Figure 35. Event Monitor Tool, db2eva - Monitored Connections

```
------------------------------------------------------------
                            EVENT LOG HEADER
  Event Monitor name: DB2STAT
  Server Product ID: SQL02011
  Version of event monitor data: 2
  Byte order: BIG ENDIAN
  Size of record: 76
  Codepage of database: 850
  Country code of database: 1
  Server instance name: db2a
------------------------------------------------------------


1) Database Header Event...
  Database Name: TPCD
  Database Path: /usr/data/db2a/SQL00002/
  First connection timestamp: 07-22-1996 17:45:34.845570


2) Event Monitor Start Event...
  Start time: 07-22-1996 17:53:43.015381


3) Connection Header Event...
  Application Id: *LOCAL.db2a.960722224534
  Sequence number: 0001
  DRDA AS Correlation Token: *LOCAL.db2a.960722161132
  Authorization Id: DB2A
  Execution Id: db2a
  Application Program Name: db2bp_32
  Client NNAME:
  Client product Id: SQL02011
  Client Database Alias: tpcd
...
```

Figure 36. Partial Output from db2evmon

If you need to find further details about a particular connection, then you can open the individual connections as data elements either by using the Selected menu options or double-clicking the connection you wish to view.

The data elements view shows details such as the client's operating system, the code page used and the version of the client DB2 product. Details about the application include connection information, buffer pool and sort details as well as database cache and catalog information. If you are not sure about any of the data elements, you can double-click any of the elements, and the DB2 help window will be displayed showing information about the element plus details on its usage and actions that may be taken if the value of the element is not acceptable.

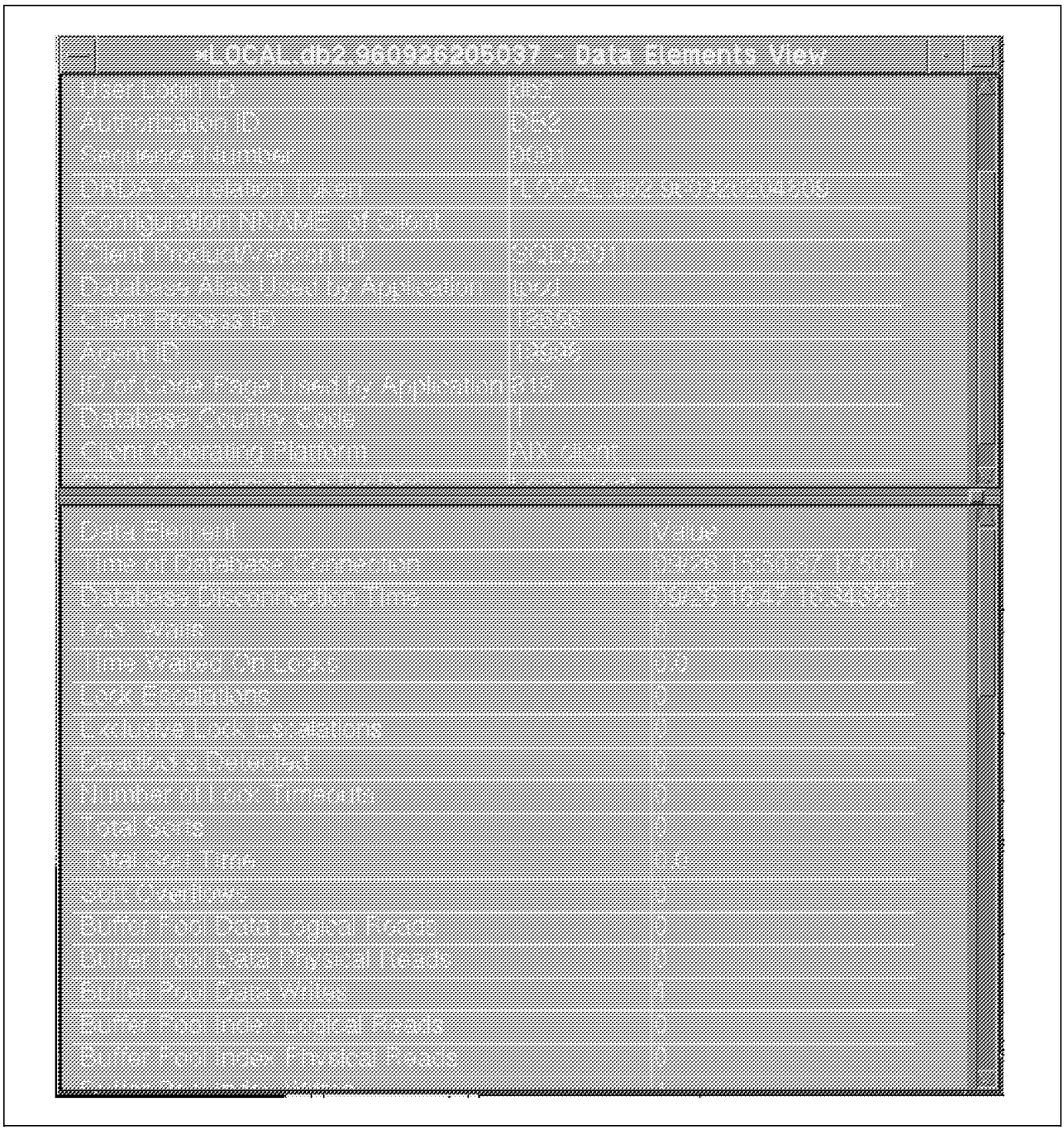


Figure 37. Event Monitor Tool, db2eva - Connection Data Elements View

4.3.2.1 Choosing Snapshot Monitor or Event Monitor

You should be aware of the differences between snapshot monitor and event monitors so that you can select the suitable tools to monitor data.

Table 3 on page 71 shows some differences between the snapshot monitor and event monitor:

	Snapshot Monitor	Event Monitor
Record Content	Current state at a particular instance in time	State at the time when the specific event occurs
Data Collection Level	At the database manager and database levels	For a single database
Usage	Sampling long-running transactions	Historical comparison monitor of transient events
Resettable Counters	Can be reset	Cannot be reset without stopping the monitor

Table 3. Differences between Snapshot Monitor and Event Monitor

Snapshot Advantages:

The snapshot monitor is especially useful to display the current status of your system and recent trends. It gives you a lot of information on database managers, databases, database connections, tables and tablespaces. It allows you to calculate the average or peak loads for given periods of time.

Event Monitor Advantages:

The event monitor allows you to collect summary information about transient events such as deadlocks and transaction completions, which is difficult to monitor through snapshots.

Snapshot monitor is used to tune configuration parameters, such as `maxagents` and `maxcagents`, which are related to applications. It also lets you check lock escalations and use of indexes, efficiency of prefetchers, etc.

The event monitor is used to plan administration, prepare maintenance, and check data placement and CPU usage, etc.

4.4 Performance Monitor

The performance monitor is a graphical tool which allows you to collect and view performance information for your database environment.

4.4.1 What is Performance Monitor

The performance monitor provides you with an interface for the collection of performance data, plus the capability to view or analyze the data collected. With this information, you will be able to further tune the database manager and database configuration parameters, diagnose any performance problems, analyze performance trends and identify application performance problems by looking at how databases utilize the system resources.

You can access the performance monitor through the Database Director's graphical user interface. The Database Director and performance monitor provide you with on-line help, including a section on getting started.

4.4.2 Using Performance Monitor

With performance monitor, you can view performance data collected by the snapshot monitor or the event monitor in real-time. The information can be displayed in tables or as graphs. It is also possible to set threshold limits with the performance monitor for the different data elements, causing a user-defined action to occur if these thresholds are exceeded.

4.4.2.1 Monitoring Snapshot Data

Using performance monitor, you can monitor snapshot data at the database manager instance, database, tablespaces, tables or connection level.

The snapshot monitoring functions include:

- Setting the intervals between snapshots
- Creating your own performance variables
- Defining threshold values and alerts

You can display monitored information using three different types of views:

- Summary

  This includes key performance variables for all the monitored objects.

- Detail

  This includes comprehensive performance data for a single monitored object.

- Graph

  This includes real-time data of current and recent activity.

To invoke the performance monitor from the database monitor, you need to select the object you wish to monitor. For example, you may want to monitor the tablespaces in the TPCD database. You need to expand the group until you see the required object. Then, using the right button, you select **start monitoring**.

After the steps above, you will get a screen similar to Figure 38 on page 73 below. Note that there is a traffic light beside the TPCD.USERSPACE1 icon. When the light turns green, it shows that you have started snapshot monitoring.

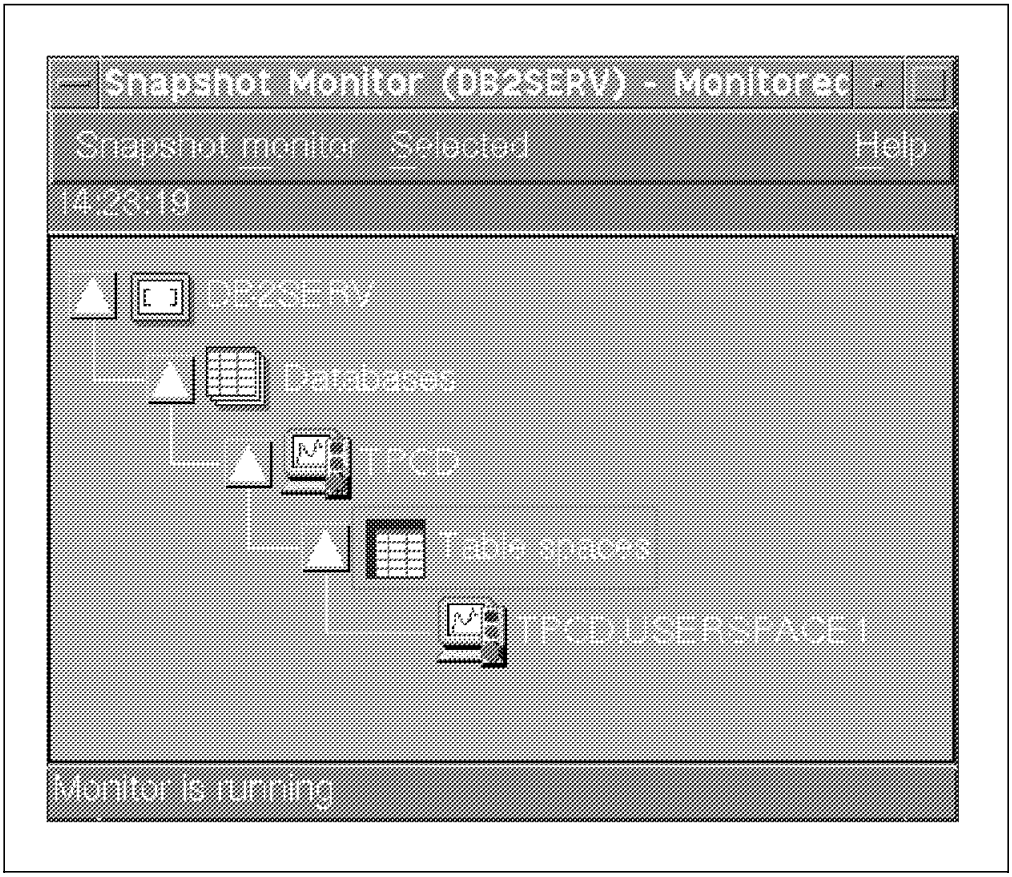


Figure 38. Performance Monitor - Tablespace Monitoring

If you want to monitor detailed information, just click on the **TPCD.USERSPACE1** icon using the right button, then select **Open as** and finally, **Performance Details**. Figure 39 on page 74 below shows an example of the performance detail screen.

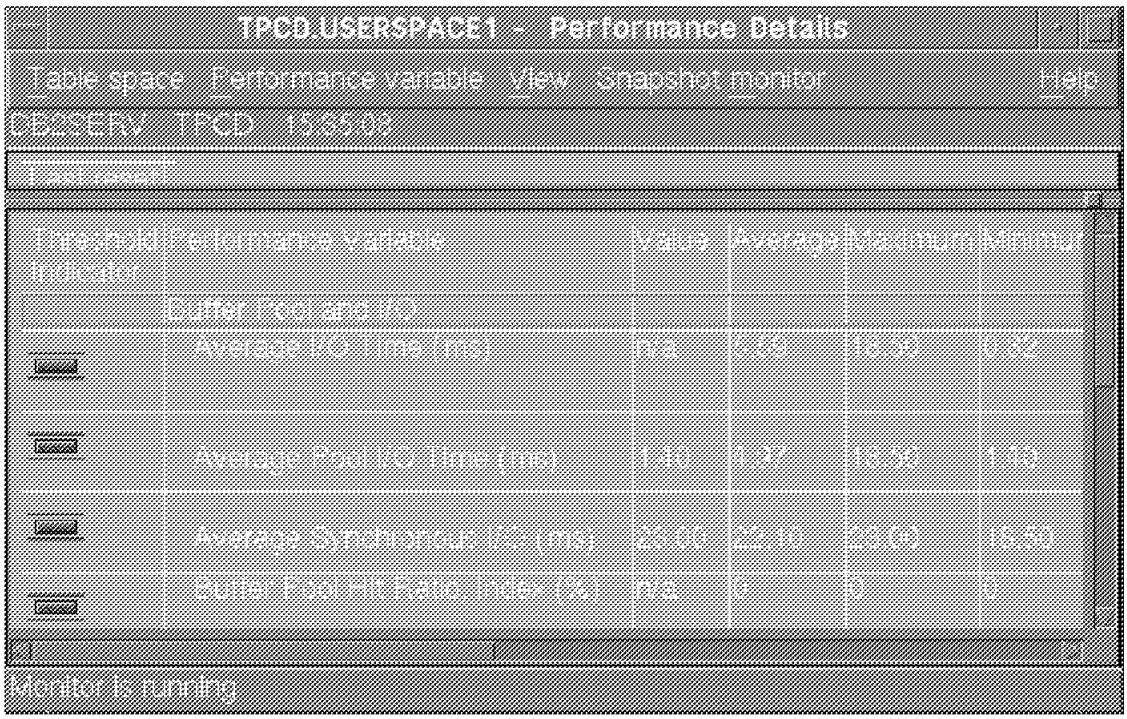


Figure 39. Performance Monitor - Tablespace Detail

Thresholds: The first column by default will be the threshold indicator; this indicator will turn red when the associated threshold value is exceeded.

You can change your threshold settings as well as define the alert action that will be performed if the value is exceeded. To set the threshold, navigate to the details screen for the object you wish to set up thresholds for, then select the data element you wish to change. Click on the **Performance variable**, then select **change threshold**. Figure 40 is an example of the threshold screen for the Average I/O Time database element.

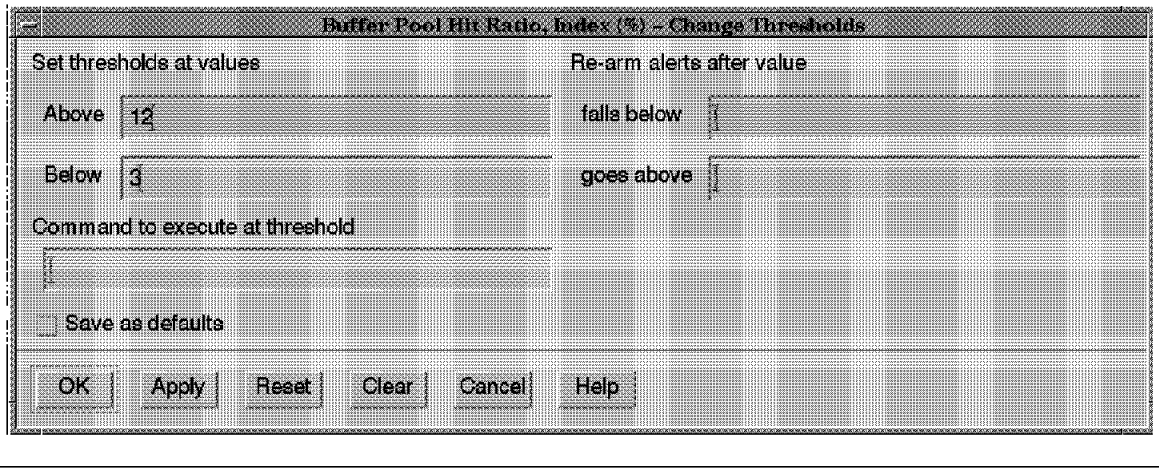


Figure 40. Average I/O Time (ms) - Change Threshold

The final method for displaying performance data is to use the graphical display. This will graphically plot snapshot data in real-time. The period of time between snapshots is configurable, as are the types of data elements you wish to display.

It is also possible in the graph screen to set your thresholds, change your X-axis and define alerts. Figure 41 is an example of the graph screen, showing the tablespace usage.

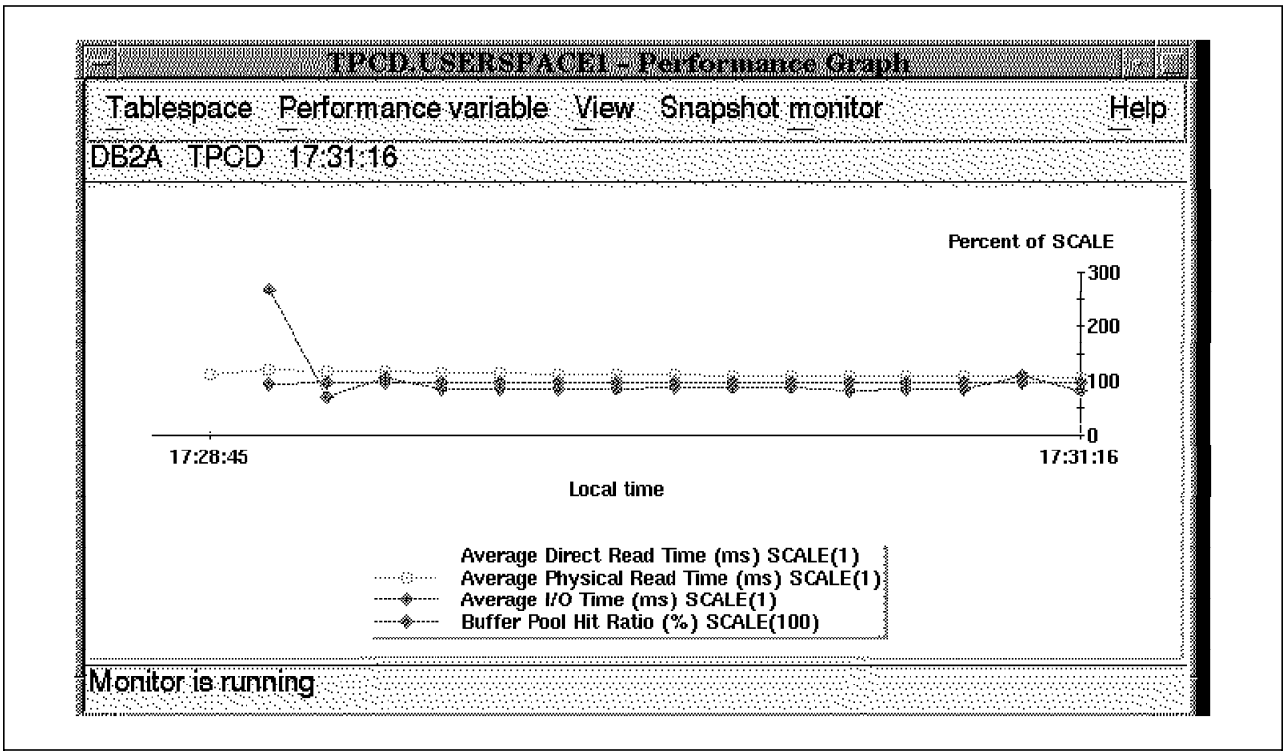


Figure 41. Performance Graph - User tablespace

To stop the snapshot monitoring, just click on the monitored object's icon and, using the right button, select **stop monitoring.**

4.4.3 Tuning with the Performance Monitor

Performance monitor can be used to display gathered information and to control the monitoring activities; it also can be used to analyze the performance of your database manager and database applications. This information can assist in the tuning of the database engine and applications.

Each DB2 instance has its own shared memory areas. The correct tuning of this shared memory will have a large impact on the performance of the database environment. The parameters that need to be looked at include:

- Buffer Pool Size *(buffpage)*
- Database Heap *(dbheap)*
- Catalog Cache Size *(catalogchche_sz)*
- Log Buffer Size *(logbufsz)*
- Utility Heap Size *(util_heap_sz)*
- Default Backup Buffer Size *(backbufsz)*
- Default Restore Buffer Size *(restbufsz)*

- Maximum Storage for Lock Lists *(locklist)*

The first parameters that can generally be tuned are for the buffer pool size, log buffer size and the lock list. The following table lists the data elements associated with these parameters and the monitor type and level that can be used to capture the information.

By viewing the database performance details screen, we can display the buffer pool hit ratio and the buffer pool hit ratio index. An example of this screen is shown in Figure 42 on page 77.

The bufferpool hit ratio indicates the percentage of time that the database manager was able to load a page from buffer memory, rather than from disk, in order to service a page request. The higher the buffer pool hit ratio, the lower the frequency of disk I/O. By monitoring this percentage, you will be able to obtain the best buffer size for your environment. You will reach a point where increasing the buffer size no longer increases performance but in fact may reduce performance by using up too much of the available system memory.

The index buffer pool hit ratio is similar to the buffer pool hit ratio, only it refers to index pages rather than data pages. Creating more index buffer pool space may also improve performance, but the same rules about monitoring the hit ratio applies here. Making this buffer pool too large may no longer give performance gains, and it may have the opposite affect.

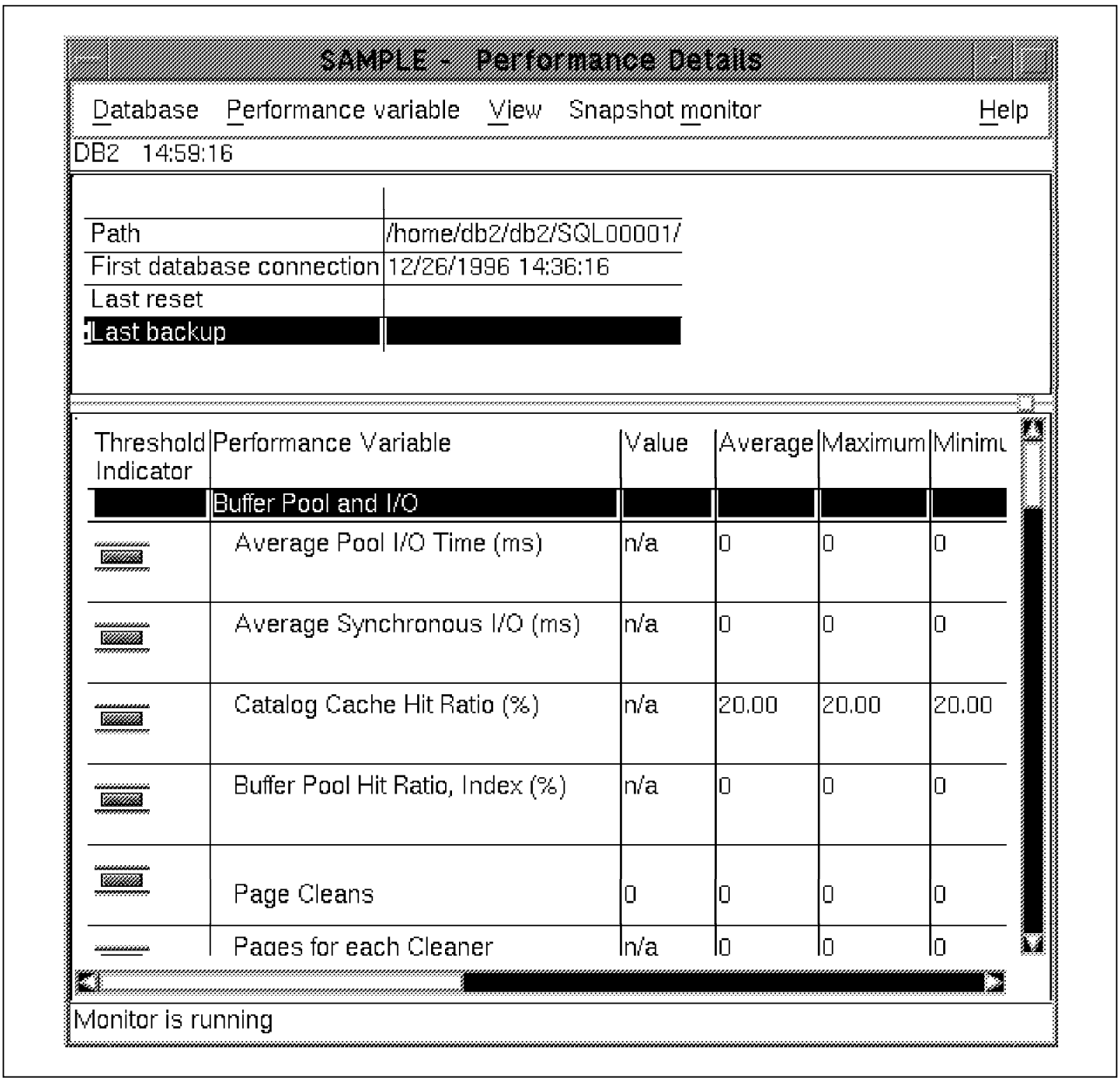


Figure 42. Performance Monitor - Database Details

To list the current settings of the buffer pool sizes, you may either use the Database Director tool or issue the following command from the DB2 Command Line Processor:

```
get database configuration for tpcd
```

A fragment of the output from this command is shown in Figure 43 on page 78.

```
Catalog cache size (4KB)              (CATALOGCACHE_SZ) = 64
Log buffer size (4KB)                        (LOGBUFSZ) = 8
Utilities heap size (4KB)                (UTIL_HEAP_SZ) = 5000
Buffer pool size (4KB)                       (BUFFPAGE) = 5000
Max storage for lock lists (4KB)             (LOCKLIST) = 100

Sort list heap (4KB)                         (SORTHEAP) = 2048
SQL statement heap (4KB)                     (STMTHEAP) = 2048
Default application heap (4KB)             (APPLHEAPSZ) = 128
Package cache size (4KB)                   (PCKCACHESZ) = 36
Statistics heap size (4KB)               (STAT_HEAP_SZ) = 4384
```

Figure 43. Database Configuration Listing (fragment)

When modifying these parameters, it is important to consider the total memory installed in your database server platform and the memory requirements of any other application running on the server. Allocating too many resources may impact any other applications on the platform.

Chapter 5. DB2 Tools and Utilities

DB2 Common Server provides a variety of tools that can be used to assist in tuning your database environment and applications. This chapter discusses each tool and its usage. Tools include:

- Explain tools based on analyzing packages
- Explain tools based on explain tables
- A benchmarking tool
- A bind file dump tool
- A tool to assist in building replica databases
- A tool to limit the resources used by applications
- An SNMP subagent with the RDBMS MIB

Not all of these tools are available on every DB2 Common Server platform. Some of these tools have a graphical user interface, such as Visual Explain, while others are ASCII tools or batch mode tools, more suitable for non-interactive execution. Finally, some of the tools are provided on an as-is basis and may not be well-documented elsewhere.

These tools will help the database administrator to:

1. Obtain the access plan for any SQL statement, static or dynamic
2. Obtain statistics on tables and indexes
3. Measure the response time of SQL statements
4. Limit the use of resources by applications

5.1 Explaining SQL Statements

The database manager retrieves information from a database when an application requests it. When several tables or several operations are involved, the database manager will have to determine an efficient step-by-step method to retrieve the required data. This step-by-step method is referred to as the access plan.

The access plan for any static SQL statement is contained in a package. The package contains the access plans for all SQL statements in the application program, or program module. If a program is made up of several program files, then each file containing static SQL will have its own package. A package is created when the application is precompiled and is stored in the system catalog tables when bound to a database. For dynamic SQL statements, the access plan for each SQL statement is created when the application is executed.

One of the most common reasons for poor performance is an inadequate choice of an access plan for SQL statements. Access plans are chosen by DB2, and the choice is influenced by the SQL statement itself, indexes, statistics, optimization level, table design and the way that data was loaded.

There are several possible causes for an inadequate access plan. The first is that the database table statistics may be out of date, or existing indexes may not be the most adequate. The SQL statements may not be well coded, or there

may be concurrence problems due to locks or isolation levels. The optimizer in DB2 Version 2.1.1 will be able to help in many of these areas, but not all problems can be caught during optimization.

Database table statistics help to define the data model to the optimizer, which in turn will determine the access plan. The statistics include information such as the number of pages in a table, the number of rows in a table, the degree of clustering of an index, the number of index levels and the number of distinct values in a column. Statistics are only gathered when explicitly requested, such as through the `runstats` utility, and are stored in the system catalogs.

Explaining SQL statements may help to determine why a program or a query is not performing as expected. The SQL statement explanation shows the access plan used to retrieve the data in response to the query. The easiest way to find out the access plan is to ask DB2. DB2 provides the explain facility to show what access plan it intends to use when the query is executed.

DB2 Common Server provides several tools to explain SQL statements and access plans:

- Tools based on analyzing packages

  Two tools are provided: `db2expln` for static SQL statements and `dynexpln` for dynamic SQL statements.

- Tools based on explain tables

  Explain table information can be retrieved by these methods:

  - Querying the tables
  - `db2exfmt` for non-graphic environments
  - Visual Explain for graphic environments

5.1.1 SQL Explain Tool

The SQL explain tool, `db2expln`, is a package analyzer, so it will only explain the access plan for a static SQL statement. A package contains the access plans for all SQL statements in the application program, or program module. The input for `db2expln` is not the explain tables but the access plan. `Db2expln` describes the access plan selected for SQL statements in the packages. It provides the step-by-step procedure used by the database manager to execute the SQL statement.

Figure 45 on page 82 shows the process involved in generating the package starting with a C application program (other supported languages are cobol, fortran and C++). The process of generating the package for the application program is called binding. The package is generated by the precompiler. The precompiler will also generate a pure C language (or cobol, fortran, or C++) file and, optionally, a bind file. Notice that, depending on the options used to invoke the `prep` command, the precompiler will generate either:

- A package
- A bind file
- A package and a bind file

Packages and Bind Files: A package is used as input for the db2expln tool. It is important to remember that if statistics are changed, packages may need to be rebound, as the optimizer determines the access plan for each SQL statement based upon the statistics available when the application was bound.

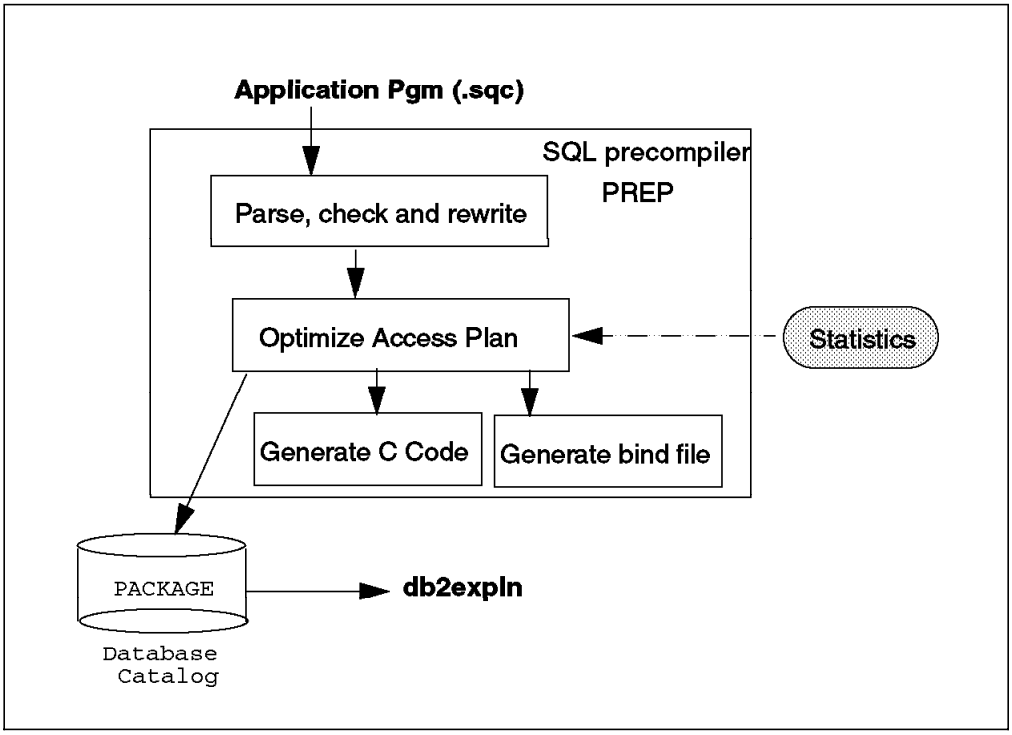


Figure 44. Creating Packages - PREP

Packages can be added or re-created using the bind command. The bind command uses a bind file as its input. The rebind commands allows you to recreate a package but does not require the bind file. Package re-creation is recommended when the statistics are changed. Figure 45 on page 82 shows the bind and rebind process.

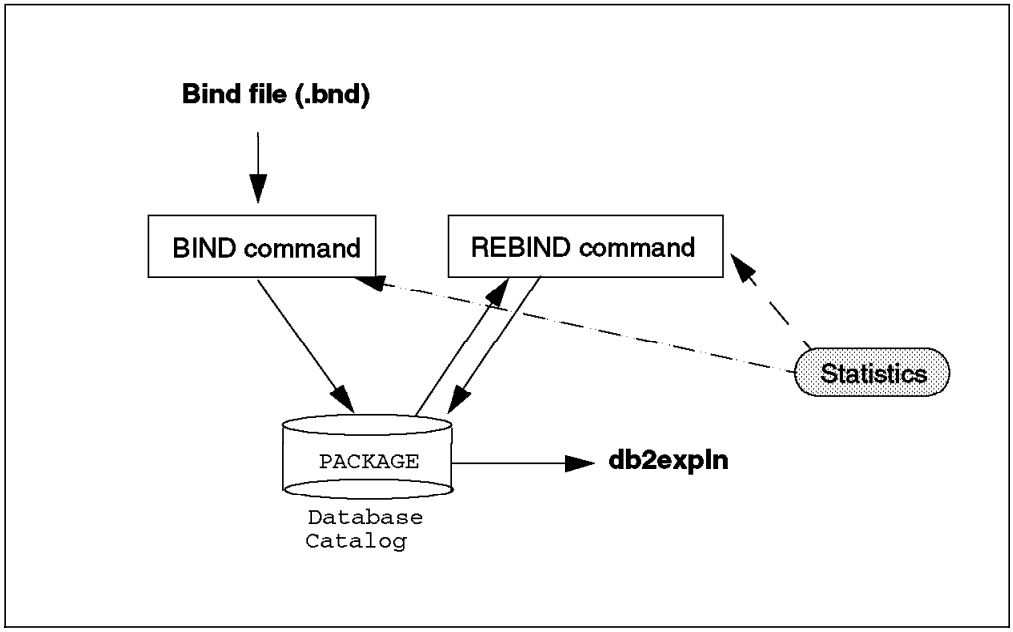


Figure 45. Adding or Replacing Packages - BIND and REBIND

In summary, a package contains the access plans for the static SQL statements for an application. The precompiler (invoked through the prep command), may create any combination of packages, bind files, or packages and bind files. The precompiler will have to be invoked every time the source program is modified, such as if an SQL statement is added to program.

The bind command uses the bind file, generated from the precompile, as its input. It can add or re-create a package. The bind command can be used to change any bind options, such as the isolation level. When re-creating a package, and thus, the access plans for the SQL statement, it will take advantage of the statistics currently stored in the system catalogs.

The rebind command does not require a bind file, as it can re-create a package from the existing package. However, no bind options can be modified. The rebind will take advantage of current database statistical information. It is the fastest way to rebind a package.

As any other application, db2expln has to bind to the database. Db2expln binds itself to the database the first time it is invoked. To execute the db2expln tool, select access to the system catalogs and bind authority is required.

Explain Output: The output of the db2expln tool consists of two parts:

- Package information

  The package information includes the package name, bind timestamp, isolation level, blocking state and optimization class. The following is a sample of the package information provided by db2expln:

```
******************** PACKAGE ****************************************

Package Name = NULLID.DB2LOOK
        Prep Date = 1996/02/02
        Prep Time = 17:38:07:085

        Bind Timestamp = 1996-06-18-14.29.49.522332
```

```
Isolation Level          = Uncommitted Read
Blocking                 = Block All Cursors
Query Optimization Class = 5
```

- Section information

  The section information includes the SQL statement being explained and the explain output of the access plan for this statement. There can be as many sections as SQL statements that are part of the package. Remember that if an application is compiled from multiple source files, then a package will exist for each source file, and that package will contain the access plans only for the SQL statements found in that source file related to that package. Db2expln may show one section or all sections of the package. This is an example of information provided for one section by db2expln:

```
Section = 3


SQL Statement:

  SELECT name, colno, coltype, length, nulls, colcard, high2key,
         low2key, length(high2key), length(low2key), avgcollen,
         nmostfreq, nquantiles, codepage
  FROM SYSIBM.SYSCOLUMNS
  WHERE tbname = :table_name and TBCREATOR = :table_creator
  ORDER BY colno


Access Table Name = SYSIBM.SYSCOLUMNS  ID = 3
|  #Columns = 14
|  Index Scan:  Name = SYSIBM.IBM01  ID = 1
|  |  #Key Columns = 2
|  |  Data Prefetch: None
|  |  Index Prefetch: None
|  Lock Intents
|  |  Table: Intent None
|  |  Row  : None
|  Create/Insert Into Sorted Temp Table  ID = t1
|  |  #Columns = 12
|  |  #Sort Key Columns = 1
|  |  Sortheap Allocation Parameters:
|  |  |  #Rows     = 2
|  |  |  Row Width = 127
|  |  Piped
Sorted Temp Table Completion  ID = t1
Access Temp Table  ID = t1
|  #Columns = 12
|  Relation Scan
|  |  Prefetch: Eligible
End of Section
```

Notice that the SQL statement shown may not exactly match the statement in the SQL application. This is because the optimizer may have re-written the statement to obtain better performance. It is this optimized statement that is shown.

The explain text is divided into steps. Major steps are left-justified. Indentation bars provide the scope of the operations. The explain may include information about:

- Table access

  Shows the name and type of the table. It also describes the access to the table. (It includes the number of columns, scan direction, row access method and lock intents.)
- Temporary tables

  Temporary tables created during the statement execution.
- Joins

  Shows the join and type of join operations performed.
- Data streams

  Shows the flow of data from one operation to the other within the access plan.
- Insert, update and delete statements
- Row identifier preparation

  Preparation of RIDs before the table is accessed. RIDs may be sorted or duplicates removed.
- Aggregation
- Miscellaneous statements

  These include DDLs, SET statements, DISTINCT clauses, and UNION operators.

The different parts of the explain output are explained in the follow sections.

5.1.1.1 Explain Text for Regular Tables

db2expln shows the names and the types of the tables being accessed. It also shows and explains access plans for temporary tables. In our previous example, two tables, a regular table and a temporary table, were explained.

```
Access Table Name = SYSIBM.SYSCOLUMNS  ID = 3
|  #Columns = 14
...
Access Temp Table  ID = t1
|  #Columns = 12
...
End of Section
```

The information provided includes the fully qualified name of the regular table. The ID shown for regular tables corresponds to the TABLEID column in SYSCAT.TABLES. For temporary tables, the values of ID are assigned by db2expln.

Indented information below the Access Table Name or the Access Temp Table statement explains how the data is being accessed. In our example, access to the regular table is listed below:

```
Access Table Name = SYSIBM.SYSCOLUMNS  ID = 3
|  #Columns = 14
|  Index Scan:  Name = SYSIBM.IBM01  ID = 1
|  |  #Key Columns = 2
|  |  Data Prefetch: None
|  |  Index Prefetch: None
|  Lock Intents
|  |  Table: Intent None
```

```
|  |  Row  : None
|  Create/Insert Into Sorted Temp Table  ID = t1
...
```

From this output, we can determine the following:

1. The table being accessed is SYSIBM.SYSCOLUMNS.
2. The number of columns being used is 14.
3. More than one row will be accessed. We know this because, if only a single row is accessed, the explain text would include the statement:

   ```
   Single Record
   ```

4. The isolation level used for this table access is the same as the isolation level for the package. The isolation level for the package is shown in the package information (uncommitted read, in our example). The statement that appears when the isolation level is different from that of the package is:

   ```
   Isolation Level: xxxx
   ```

5. The scan direction is forward, which is the default. If the database manager reads the rows in reverse order, the following text will be shown:

   ```
   Scan Direction = Reverse
   ```

6. Rows are being accessed through an index. See "Row access."
7. No locks will be acquired. See "Lock intents" on page 86.
8. A sorted temporary table, if necessary, will be created.

Row access: There are two ways of accessing data in a table. The first is by directly reading the table (relation scan), and the second is accessing an index on the table (index scan). The db2expln utility shows which of these two methods is used to access the data.

If a table scan is being used, the following statement will be displayed:

```
Relation Scan
Prefetch: Eligible
```

The Prefetch statement displays if prefetch is applicable.

If an index scan is used, the format of the statement will be:

```
Index Scan:  Name = schema.name ID = xx
```

where ID is the IID (index identifier) column in SYSCAT.INDEXES.

Index scans are further explained through additional statements within the explain output. From our example, the statements that explain the index scan are:

```
|  Index Scan:  Name = SYSIBM.IBM01  ID = 1
|  |  #Key Columns = 2
|  |  Data Prefetch: None
|  |  Index Prefetch: None
```

From these additional statements, we know how the index scan will be performed:

1. Two columns in the index key are being used to delimit the index scan range. If the value of #Key Columns was 0, a full scan of the index would be performed.
2. Table data is being accessed. This is known because, when only indexes are accessed, the following statement appears:

```
Index-Only Access
```

3. No data or index prefetch will be used.
4. No rows will be accessed by row ID (RID). If rows are accessed by RID, the following statement is displayed:

```
Fetch Direct
```

Lock intents: Lock intents show the type of lock that will be acquired at both the table level and the row level. The statements from our example that explain the lock intents are:

```
|  Lock Intents
|  |  Table: Intent None
|  |  Row  : None
```

Possible values for table and row locks are:

- Table lock
  - Exclusive
  - Intent Exclusive
  - Intent None
  - Intent Share
  - Share
  - Share Intent Exclusive
  - Super Exclusive
  - Update.
- Row lock
  - Exclusive
  - None
  - Share
  - Update

5.1.1.2 Explain Text for Temporary Tables

Temporary tables are used when intermediate results do not fit in the sort heap, or if subqueries are used in the search condition. It is also possible that the temporary tables will be sorted or unsorted, and so the access of these tables can affect performance.

The access plan provides a temporary table even if it intends to ‘pipe’ the sort. By doing this, the database manager will be able to know how to build the temporary table if it’s needed at execution time, and the access plan will be independent of whether the results are kept in the sort heap or placed in the temporary table. In our example, one sorted temporary table was explained. The statements included in the explain text were:

```
|  Create/Insert Into Sorted Temp Table  ID = t1
|  |  #Columns = 12
|  |  #Sort Key Columns = 1
|  |  Sortheap Allocation Parameters:
|  |  |  #Rows     = 2
|  |  |  Row Width = 127
|  |  Piped
Sorted Temp Table Completion  ID = t1
```

From these statements, we can see the following:

1. The temporary table is sorted. This is because, for non-sorted temporary tables, the following statement would appear:

```
Create/Insert Into Temp Table ID = tn
```

2. Rows inserted in the temporary table have 12 columns.
3. There is one key column used in the sort.
4. The sort will require 2 rows of 127 bytes. This information can be helpful when trying to pinpoint problems with sort heaps.
5. The sort is piped. This means that the database manager will try to keep the sort in the sort heap. But if there is not enough space in the heap, it will use the temporary table. So the access plan is prepared for both events!

Notice that the table access information for the temporary table is detached from this step. Explain text for temporary table access is shown in the next step. Statements used to explain temporary table access are the same as those used for 'regular' table access.

```
Access Temp Table  ID = t1
|  #Columns = 12
|  Relation Scan
|  |  Prefetch: Eligible
End of Section
```

5.1.2 Explaining Dynamic SQL

Access paths for dynamic SQL statements are not determined until the statement is being executed. In order to explain how dynamic SQL statements will access the data, the following procedure can be used:

1. Include the dynamic SQL statement in a C program as a static SQL statement.
2. Prep and bind the C program.
3. Explain the package through db2expln.

DB2 common server provides the dynexpln utility, which will perform these steps for you. It is a REXX procedure for OS/2 servers, a shell script for AIX servers and an executable program for NT servers. It will execute these three steps. The syntax of dynexpln is:

```
dynexpln <dbname> "<SQL statement>"
```

Bind options are specified through the environment variable DYNEXPLN_OPTIONS, as in the following example (AIX servers, for OS/2 servers use the set command):

```
export DYNEXPLN_OPTIONS='blocking all isolation ur queryopt 3'
```

The output for the these statements is the same as explained in 5.1.1, "SQL Explain Tool" on page 80.

5.1.3 Explain Tables

Explain tables are used by different tools to show the access plans of SQL statements. A set of explain tables should be created by each user working with these tools. Though explain tables can be queried as any other table, information from these tables can easily be retrieved and formatted by db2exfmt.

Explain tables can also store explain snapshots. Snapshots are stored as binary objects and are only used by the Visual Explain tool. An explain snapshot contains all the information required by Visual Explain to graph access plans, as

well as the values of statistics and database configuration parameters at the time the explain snapshot was taken.

5.1.3.1 Creating Explain Tables

Explain data is kept in the explain tables. These tables must be created before Visual Explain is executed. For each database, a set of explain tables need to be created. If multiple users are using Visual Explain, then multiple explain tables should be created. There are seven tables used by the explain facility:

- EXPLAIN_INSTANCE

  Contains the database configuration parameters settings when the explain snapshot was taken.
- EXPLAIN_STATEMENT

  Contains the SQL statement and the explain snapshot.
- EXPLAIN_ARGUMENT

  Contains the characteristics for each operator.
- EXPLAIN_OBJECT

  Contains all the data objects required by the access plan.
- EXPLAIN_OPERATOR

  Contains all the operators needed in the execution of the SQL statement.
- EXPLAIN_PREDICATE

  This table contains all the predicates applied by each operator.
- EXPLAIN_STREAM

  This table contains the input and the output data streams between operators and table objects.

The easiest way to create these tables is to use the supplied EXPLAIN.DDL file. The EXPLAIN.DDL file is provided under the sqllibmisc subdirectory for OS/2 or NT servers (for AIX servers, under ./sqllib/misc). To create these tables, follow these steps:

1. Start the instance though db2start.
2. Connect to the database.
3. Create the explain tables executing this command:

   ```
   db2 -tf EXPLAIN.DDL
   ```

5.1.3.2 Populating Explain Tables

Once these tables have been created, they are empty. These tables are populated with two different sets of information:

- Explain Table Information.

  Explain table information will populate all seven explain tables. The information stored in these seven tables can be queried or used as an input for the db2exfmt tool.
- Explain Snapshot Information.

  Explain snapshot information will only populate two tables. It includes the snapshot of the statement. The snapshot is stored in an internal binary format in one of the explain tables. Snapshot information is only used by Visual Explain.

Explain Table Information: Explain table information is collected in different ways for dynamic and static SQL statements. These are as follows:

1. Dynamic SQL

   To collect explain table information, there are two possible methods;

   a. An EXPLAIN SQL statement such as:

      db2 explain plan selection for <sql statement>

      where <sql statement> is the statement to be explained. Explainable SQL statements are DELETE, INSERT, SELECT, SELECT INTO, UPDATE, VALUES and VALUES INTO.

   b. Change the value of the current explain mode register so that explain table information is collected by default when subsequent SQL statements are executed:

      db2 set current explain mode <value>

      where <value> may be:

      - NO

        Subsequent dynamic SQL statements will be executed, but no explain table data will be collected
      - YES

        The statements will be executed and the explain table data collected
      - EXPLAIN

        Explain table data will be collected for the statements, but the statement will not be executed.

      Every dynamic SQL statement executed after the value of the current explain register has been changed will be affected. Care should be taken not to leave the current explain mode register to an undesired value.

2. Static SQL

   There are two different options for static SQL;

   a. Within the program, using a static EXPLAIN statement.
   b. At prep/bind time using the EXPLAIN option. Values of the EXPLAIN option can be:

      - NO

        No explain table information will be gathered.

      - YES

        Explain table information will be gathered for static SQL statements contained in the program.

      - ALL

        Explain table information will be collected for each static or dynamic SQL statement contained in the program, even if the db2 set current explain mode no command has been executed.

Explain Snapshot Information: Explain Snapshot information is stored in the EXPLAIN_INSTANCE and EXPLAIN_STATEMENT tables, while the snapshot itself is stored as a binary object in the EXPLAIN_STATEMENT table. You should note that the explain snapshot is not a part of the snapshot monitor. Explain snapshots are

taken in two different manners for static and for dynamic SQL. These are listed below and illustrated in Figure 46 on page 91:

1. Dynamic SQL
   - An EXPLAIN PLAN FOR SNAPSHOT SQL statement such as:

     db2 explain plan for snapshot for <sql statement>

     where <sql statement> is the statement object of the snapshot.
   - An EXPLAIN PLAN WITH SNAPSHOT SQL statement such as:

     db2 explain plan with snapshot for <sql statement>

     where <sql statement> is the statement object of the snapshot. The WITH SNAPSHOT option will collect both explain table information and explain snapshot information for the statement.
   - To take explain snapshots of any dynamic SQL statement by default, the following DB2 command is used to assign a value to the current explain snapshot register:

     db2 set current explain snapshot <value>

     where <value> can be:

     – NO

       No explain snapshot will be collected. Subsequent statements will be executed.
     – YES

       Subsequent statements will be executed and the explain snapshot collected for each one.
     – EXPLAIN

       Explain snapshot information will be collected for every statement, but the statement will not be executed.

     Every dynamic SQL statement executed after the value of the current explain snapshot register has been changed will be affected. Care should be taken not to leave the current explain mode register to an undesired value.
2. Static SQL

   There are two different options available for static SQL:

   a. Within the program, using the API, SQL_EXPLSNAP_OPT. when preparing/binding.
   b. At precompile/bind time using the EXPLSNAP option. Values of EXPLSNAP can be:
      - NO

        No explain snapshot will be taken.
      - YES

        An explain snapshot for each static SQL statement included in the program will be taken.
      - ALL

        An explain snapshot for each static and dynamic SQL statement included in the program will be taken. This will occur even if the db2 set current explain snapshot no command has been executed.

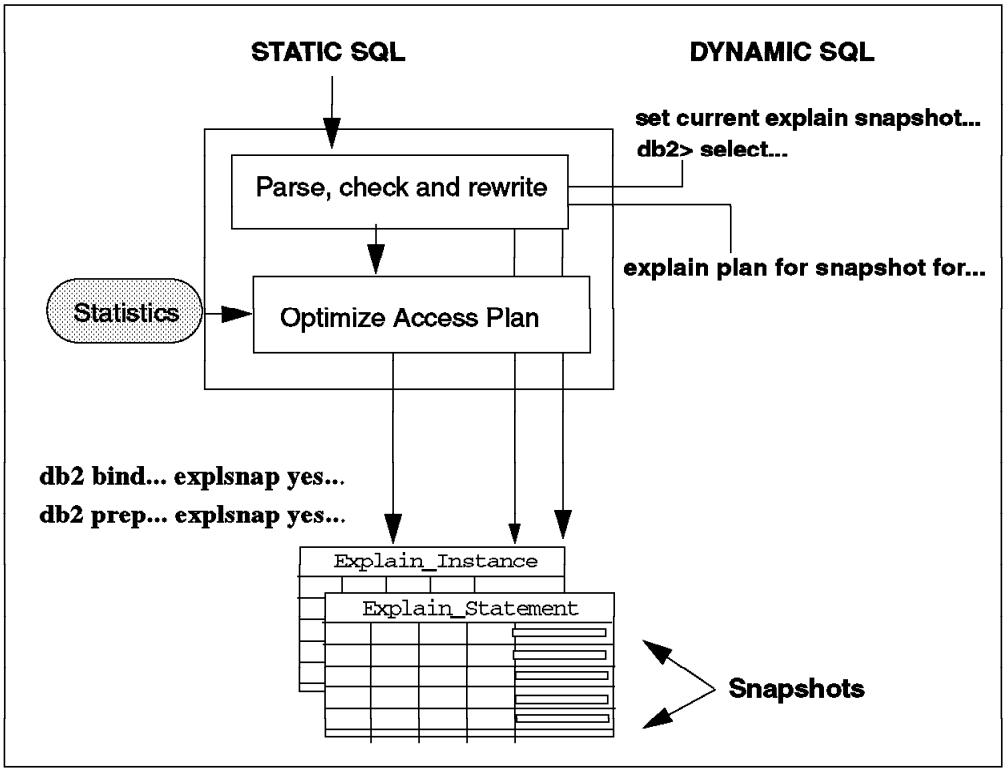


Figure 46. Populating Explain Tables with Snapshots

5.1.4 Visual Explain

Visual Explain is a graphical tool used to display and analyze access plans. It also provides information about the decision criteria used by the optimizer in choosing one plan over other alternative plans. The access plans displayed may belong to either static or dynamic SQL statements.

Visual Explain can be started through the Database Director or using the db2vexp command. When using the db2vexp command, you will have to provide the database name and the SQL command to be explained.

Visual Explain uses two concepts when dealing with SQL statements. These are explainable statements and explained statements.

- Explainable statement

  An explainable statement is a static SQL statement that is included in a package. An explainable SQL statement may or may not have associated explain information.

- Explained statement

  An explained statement is a static or dynamic SQL statement for which an explain snapshot has been taken. A graph showing the access plan for the statement can only be shown if the statement is an explained statement.

Visual Explain obtains its input information from these sources:

- The EXPLAIN_STATEMENT and EXPLAIN_INSTANCE tables.
- Database configuration parameters.

- Statistics collected by runstats or load utilities. These statistics are stored in the following tables:
  - SYSSTAT.STATS
  - SYSSTAT.TABLES
  - SYSSTAT.COLUMNS
  - SYSSTAT.COLDIST
  - SYSSTAT.INDEXES
  - SYSSTAT.FUNCTIONS

5.1.4.1 Viewing explained statements

Explained statements can be accessed through the explained statements history option in the Database Director. Selecting this option brings up a window with a list of explained SQL statements.

Every row in the table shown is an explained statement. For each of the explained statements, the following information can be provided:

- Package Name
- Package Creator
- Explain Snapshot, if any has been taken for the statement
- Latest Bind, to see if the statement is associated with the latest bound package
- Dynamic Explain, set to yes for dynamic SQL statements
- Explain Date and Time
- Total Cost, of the statement, in timerons
- Statement Number, in the source code
- Section Number, the number of the section in the package
- Query Number
- Query Tag and SQL Text (only the first 100 characters)

Notice that the information shown is configurable through the View menu.

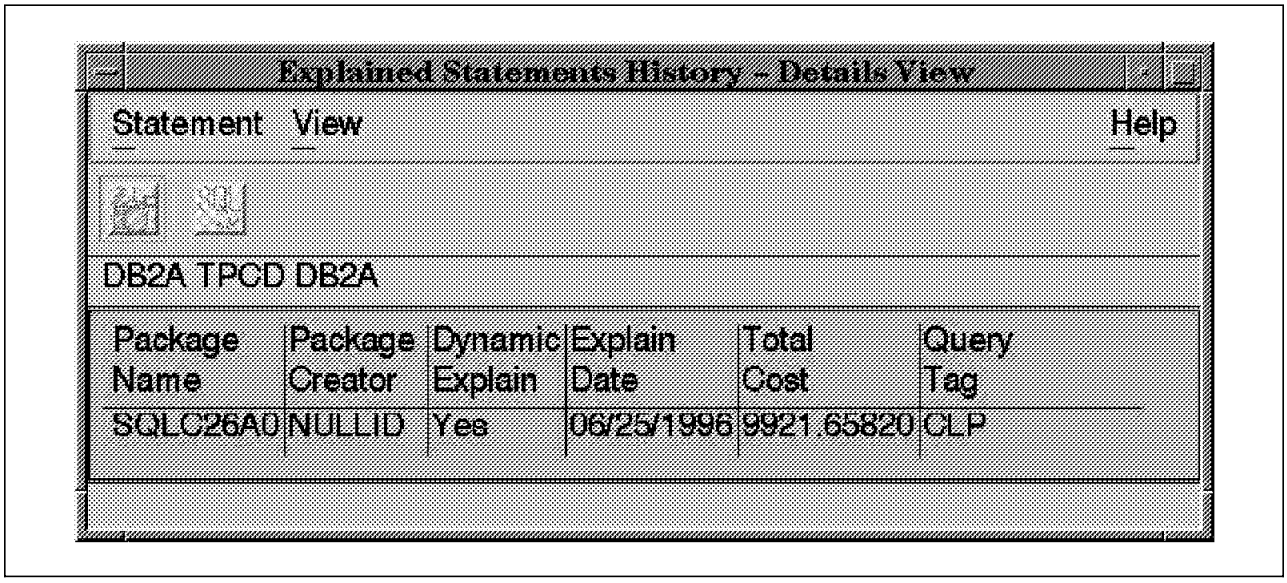


Figure 47. Explained Statements History

In Figure 47, only six columns are configured to be shown. Selecting the explained statement, several options are available:

- Show the access plan for this statement

  The access plan can be displayed through:

  - The pop-up menu brought up by the right button of the mouse.
  - Clicking on the picture camera icon
  - Through the pull-down menu: Statement, Show Access Plan.

- Show the SQL statement text

  The SQL statement text is shown by:

  - The pop-up menu brought up by the right button of the mouse
  - Clicking on the SQL icon
  - Through the pull-down menu: Statement, Show SQL Text

5.1.4.2 The Access Plan

The access plan is shown by Visual Explain as a hierarchical tree. The tree should be interpreted from bottom to top, or right to left, depending on the selected configuration. Figure 48 on page 94 shows an example of how Visual Explain represents an access plan. The steps chosen by the optimizer are shown as geometric figures. These figures or nodes may be:

- Operand nodes

  Operand nodes represent tables and indexes. A table is represented as a rectangle, and an index is shown as a diamond.

- Operator nodes

  Operator nodes are represented as octagons or hexagons. Hexagons are used for functions and octagons for operations. The following are types of operators available:

  - Delete
  - Fetch
  - Filter
  - Grpby - Groups rows
  - Insert
  - Ixscan - Index scan
  - Msjoin -Merge Join
  - Nljoin - Nested loop join
  - Return - Represents the return of data to the application
  - Ridscn - Scan a list of row identifiers
  - Sort
  - Tbscan - Table scan
  - Temp - Temporary table creation
  - Union
  - Unique - Duplicate row elimination
  - Update
  - Genrow - Built in function that produces a table

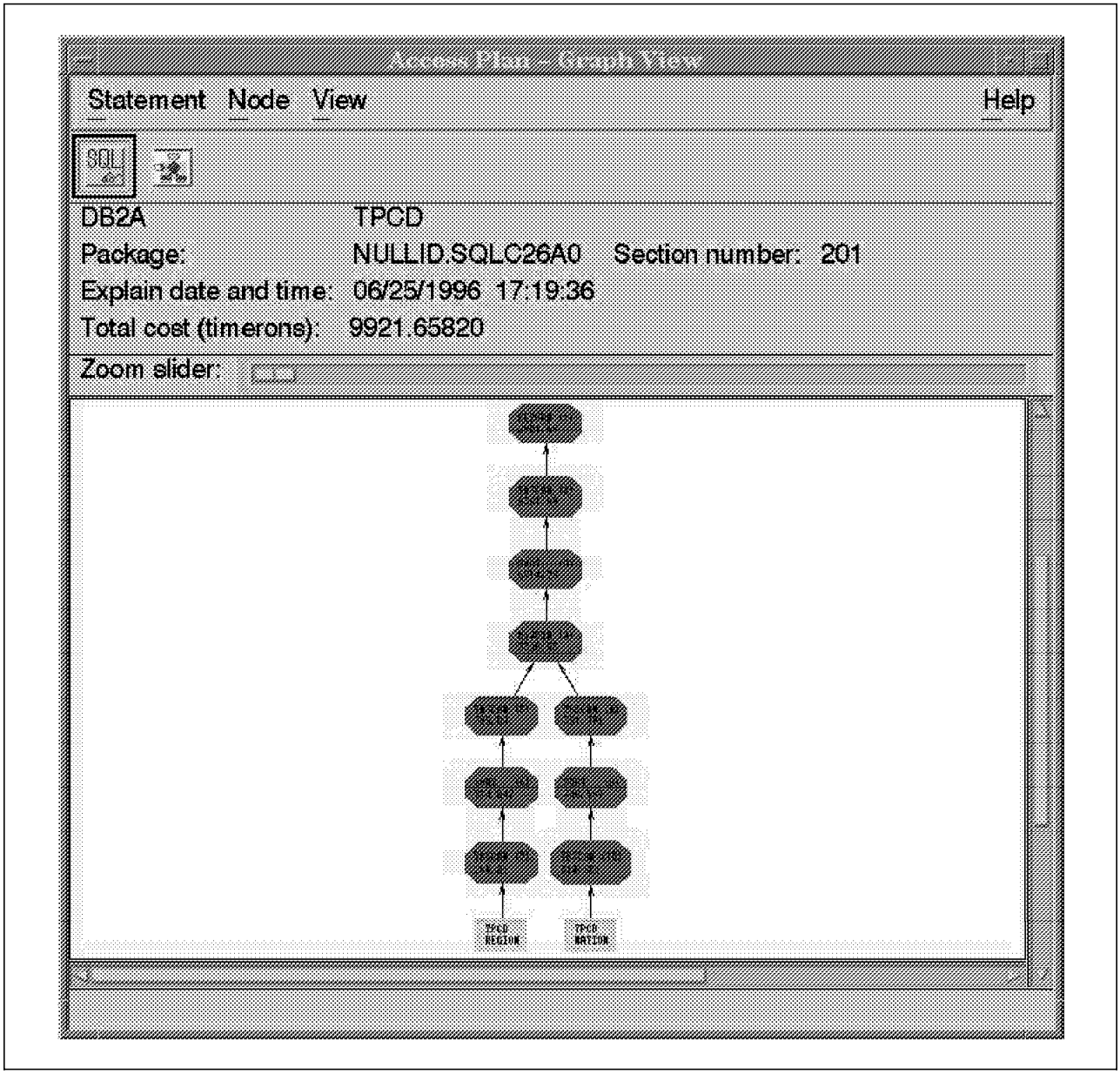


Figure 48. The Access Plan

A more detailed view of the nodes in the access graph can be obtained using the zoom slider as shown in Figure 49 on page 95. The number present in each operator node is the I/O cost, as this is what has been selected for display. The access graph can be configured to show other information on the operator nodes such as global cost, CPU cost, I/O cost or cardinality.

Global cost is measured in a unit called timerons. A timeron is an estimate of the total resources that will be consumed by an operator, accounting for both CPU and I/O activity.

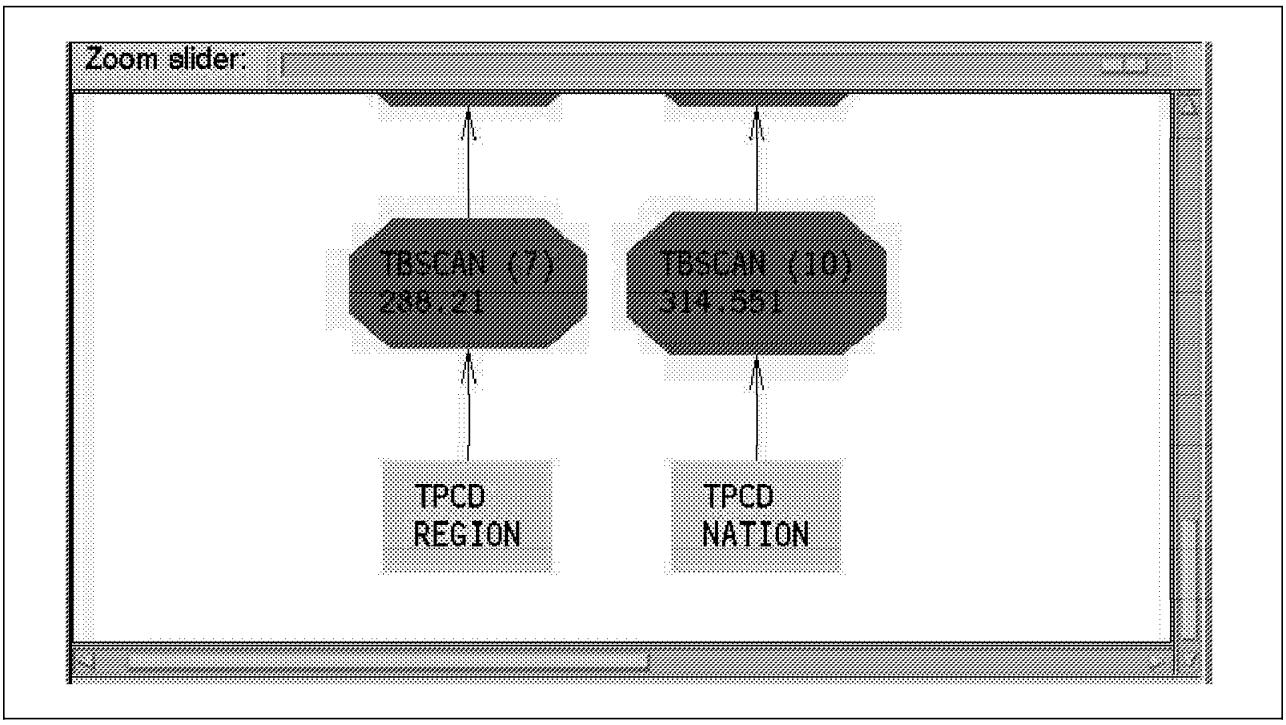


Figure 49. Access Plan Using the Slider

You can see in the access plans that nodes are linked by arrows. These arrows indicate the flow of data. In addition, each operator node is assigned a unique number, enclosed in parenthesis, for further identification.

5.1.4.3 Viewing Statistics and Details

To view the statistics of an operand (a table or an index) or to view more details of an operator, you only have to click on the node from the access plan window. The type of information that is returned depends upon the type of node selected.

Operand Nodes: Clicking on an operand node will display the statistics associated with the notes table or index, as shown in Figure 50 on page 96. This figure shows the statistics of the table TPCD.NATION and next to it, the statistics of its column. NAME is also shown.

The statistics shown compare values for the statement when it was explained with the values currently stored. Notice that the current values were obtained the last time that the `runstats` utility was executed for the table.

Table statistics include:

`CREATE_TIME`	Date of creation of the table.
`STATS_TIME`	Date when a change was made to any of the statistics for the table. Notice that in Figure 50 on page 96, the value is set to `Statistics not Updated`, which means that there is a difference between the current statistics and those used when the statement was explained.
`TABLESPACE`	Name of the tablespace where the table is stored.
`INDEX_TABLESPACE`	Name of the tablespace where the indexes of the table are stored.
`COLCOUNT`	Number of columns in the table.
`CARD`	Number of rows in the table.
`NPAGES`	Number of pages of the table that contains more than one row.

FPAGES	Number of pages used to store the table.
OVERFLOW	Number of overflow pages used by the table

Index statistics include:

CREATE_TIME	Date of creation of the index.
STATS_TIME	Date when a change was made to any of the statistics for the index.
UNIQUERULE	Shows if it is a primary index.
COLCOUNT	Number of columns in the index key.
NLEAF	Number of leaf pages.
NLEVELS	Number of index levels.
FIRSTKEYCARD	Cardinality of the first column of the index.
FULLKEYCARD	Cardinality of all columns of the index.
CLUSTERRATIO	Data clustering of the index.
CLUSTERFACTOR	Data clustering factor of the index.

Column statistics are shown by clicking on the Referenced Column button of the table/index statistics window. Column statistics include:

COLNO	Number of the column in the table.
COLCARD	Cardinality of the column.
AVGCOLLEN	Average length of the column.
NMOSTFREQ	Filled up by distribution statistics, it is the number of most frequent values (that is, duplicates) in the column. In Figure 50, it is set to -1. This is caused by not using the `with distribution` clause when `runstats` was executed.
NQUANTILES	Filled up by distribution statistics, it is the number of quantiles. Set to -1 in the example after running `runstats` without the `with distribution` clause. Used by the optimizer to figure out the distribution of data within a column.
HIGH2KEY	Second highest value in the column.
LOW2KEY	Second lowest value in the column.

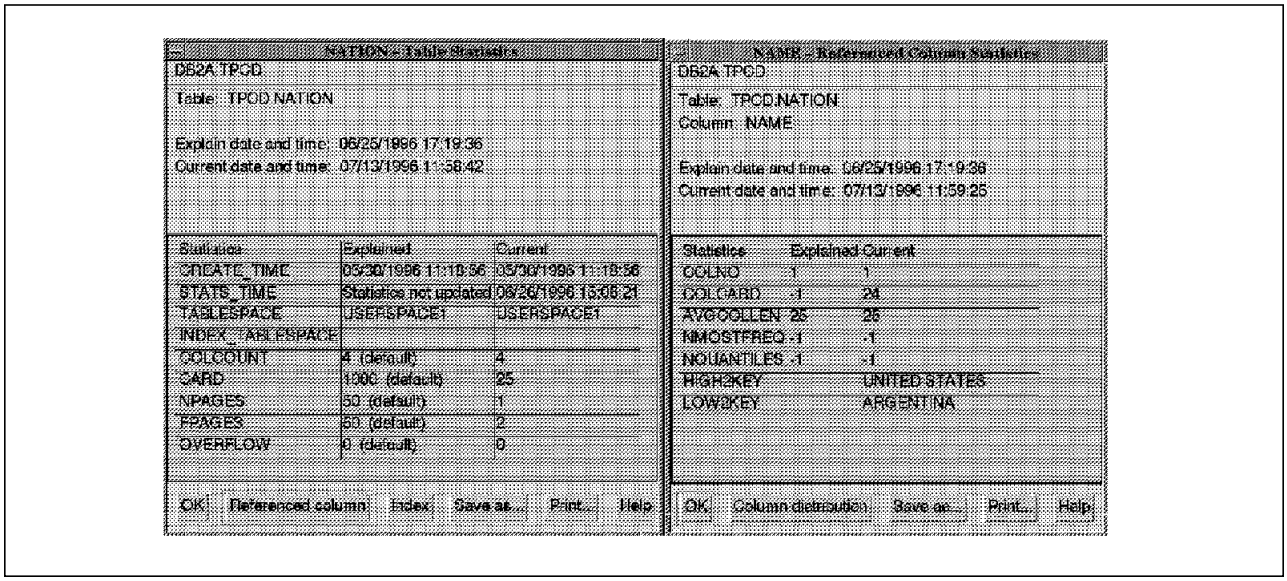


Figure 50. Statistics of an Operand

Operator Nodes: Clicking on an operator node will show the details of that operator. Figure 51 on page 98 is an example of the details for a table scan. The details of an operator are grouped into three areas:

1. Cumulative Costs
   - Total Cost
     Measured in timerons.
   - CPU Cost
     Number of instructions.
   - I/O cost
     Number of I/O operations. This includes number of seeks and page transfers.
   - First Row Cost
     Only available if a full level of detail is selected. This is not shown in Figure 51 on page 98, as the level of detail selected is overview. The first row cost is the estimated cost of obtaining the first row of the answer set.
2. Cumulative properties of the tables accessed by the operator.
   - Tables
     Name of the tables accessed by the operator.
   - Count of columns
     Number of columns accessed by the operator.
   - Order of columns
     Ascending, descending or none.
   - Count of predicates
     Set of predicates applied (searches or comparisons included in `WHERE` or `HAVING` clauses).
   - Cardinality
     Number of rows to be returned. This value is obtained from the statistics tables. The `runstats` utility should be executed after updates to maintain this value.
3. Input Arguments
   The details vary depending on the type of the operator. For a table scan, the arguments shown are:
   - Scan source
     Base tables in the example.
   - Scanned Table
     Name of the table to be scanned.
   - Columns retrieved
     Name of the columns accessed.
   - Count of sargable predicates
     A sargable predicate is a predicate that can be resolved by simple comparisons instead of by subqueries.
   - Scan direction
     Forward or reverse.
   - Count of residual predicates
   - Prefetch
     Only shown if the full level of detail is selected. Shows whether sequential detection is enabled or displays a positive number showing the number of row identifiers for list prefetching.
   - Maximum pages
     Only shown if the full level of detail is selected. Maximum pages expected to be read from disk. `None` means that prefetching is not enabled. `All` means that all pages are expected to be read, and `<nnn>` is a positive number indicating the number of pages that are expected to be read.
   - Lock intents

Lock modes for accessing the table or rows.

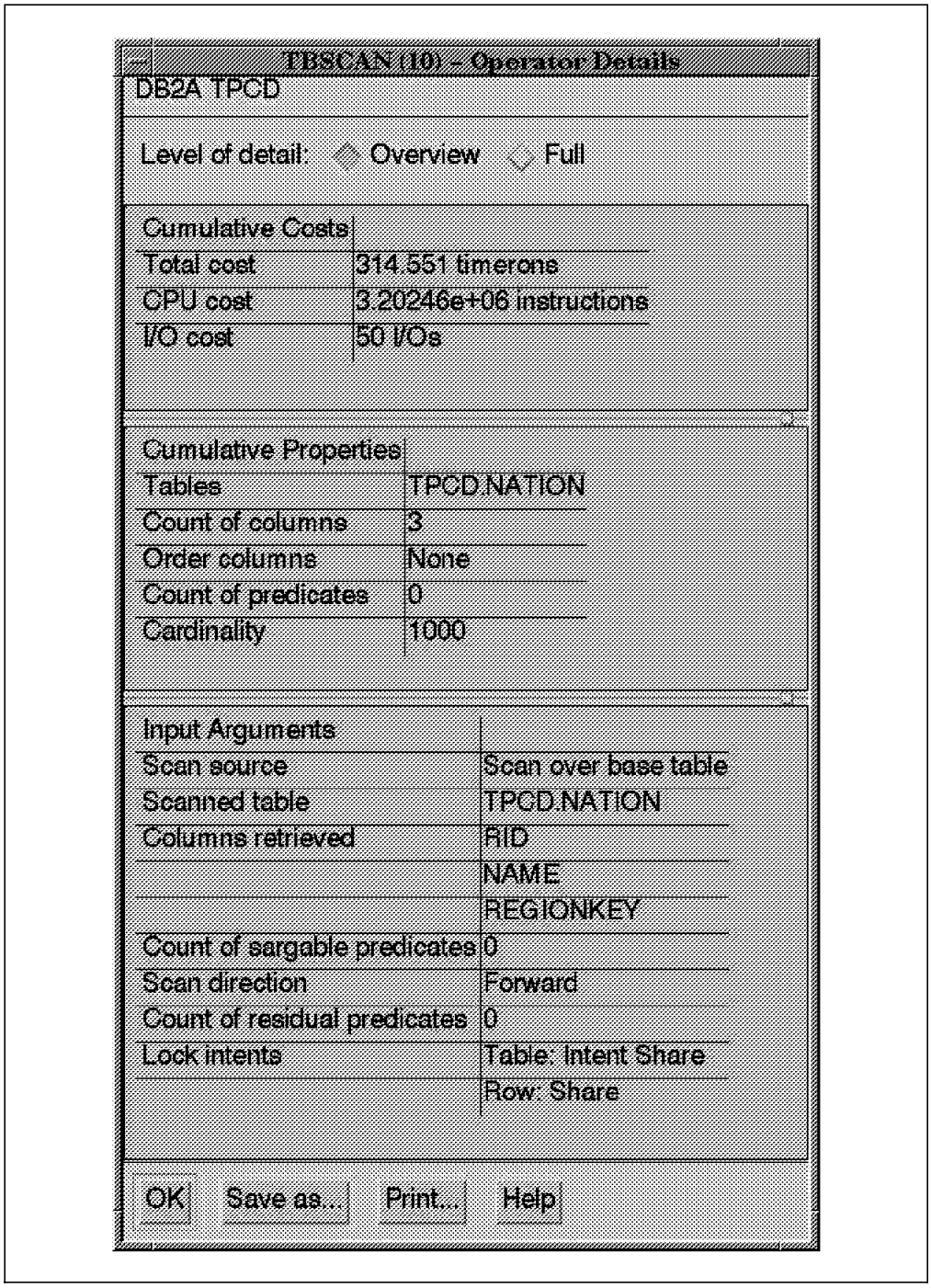


Figure 51. Details of an Operator

5.1.4.4 Optimizer

Visual Explain also provides information about the work performed by the optimizer. The optimizer can rewrite a query and generate the access plan. The access plan generated will depend upon the values of certain configuration parameters, such as the size of the buffer pool, the size of the table or the level of isolation.

Visual Explain can provide you with the information about the optimizer such as the difference between the coded SQL statements and the optimized SQL statements. Figure 52 on page 99 shows an example of an original SQL statement and the optimized version of the statement.

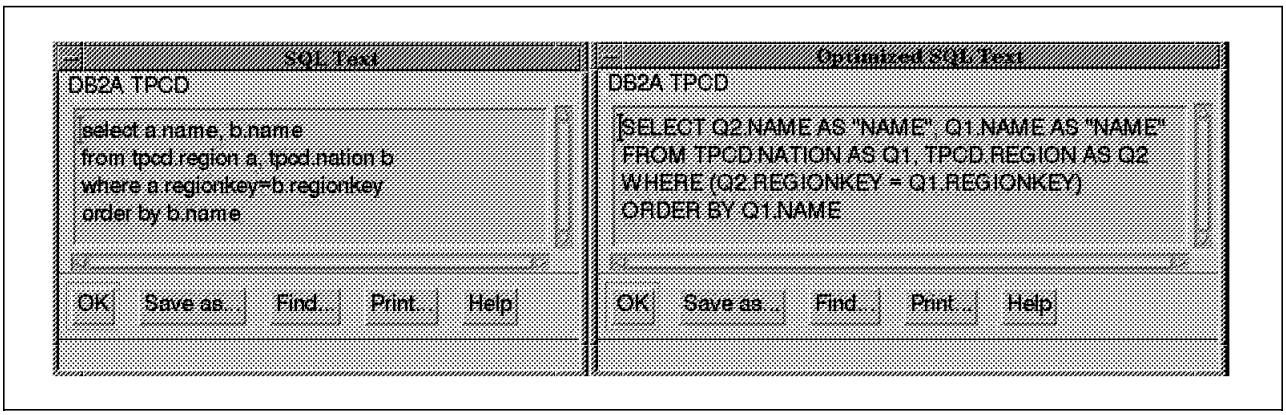


Figure 52. Optimizer - Optimized SQLStatement

Through Visual explain, you can also obtain the values of the database configuration parameters that have been used by the optimizer to calculate the cost and to determine the access plan. The values of these parameters are compared to their current values, as illustrated in Figure 53 on page 100.

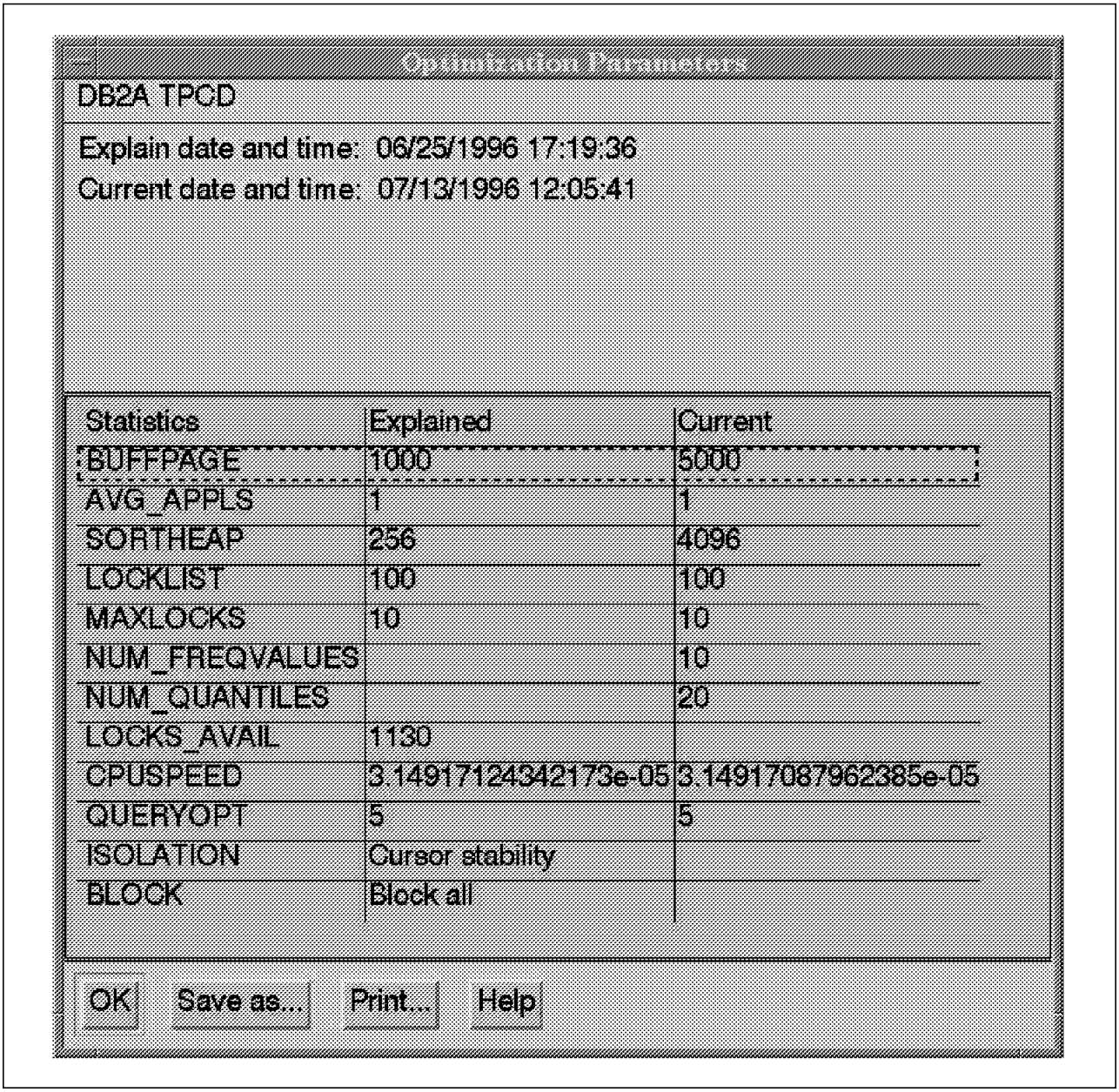


Figure 53. Optimizer - Database Configuration Parameters

5.1.5 Explain Table Formulator

The tool `db2exfmt` will put the information found in the explain tables into a format that can be more easily interpreted. It provides the access plan for the explained statements, and it obtains information from all seven explain tables, as seen in Figure 54 on page 101.

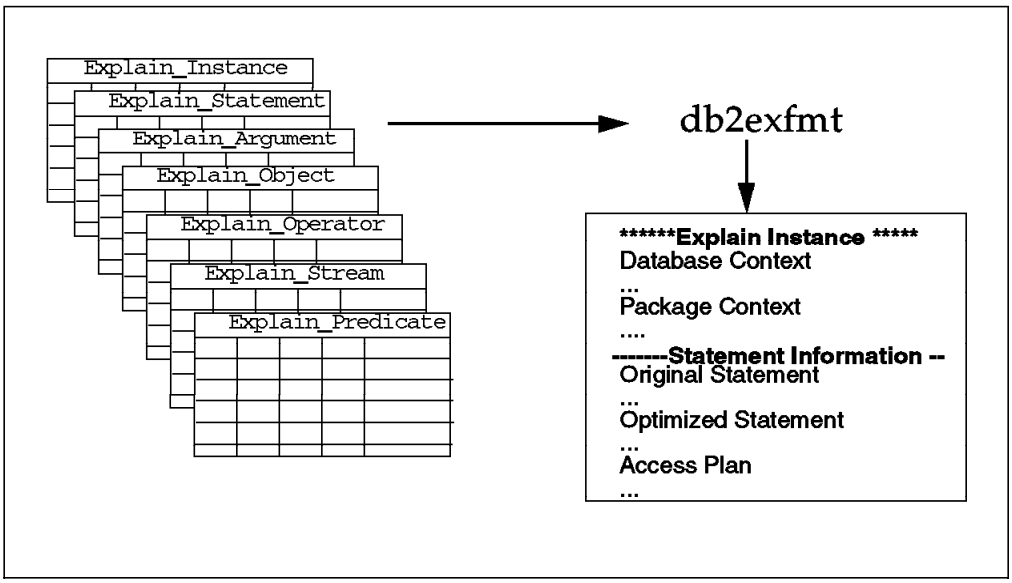


Figure 54. Db2exfmt

The explain table formatting tool will prompt you for the following information:

- Database name
- Explain timestamp
- Name and schema of the package

This information can be obtained using the Database Director. Select **Packages**, and open the package with the Explained statements history menu item. Alternatively, you can query the EXPLAIN_INSTANCE table. When db2exfmt is invoked without parameters, it will request the required information, as shown in the following example:

```
$ db2exfmt

DB2 Common Server Version 2, 5622-044 (c) Copyright IBM Corp. 1995
Licensed Material - Program Property of IBM
IBM DATABASE 2 Explain Table Format Tool

Enter Database Name ==> tpcd
Connecting to the Database.
Connect to Database Successful.
Enter up to 26 character Explain timestamp ==> 1996-06-28%
Enter up to 8 character source name ==> sqlc26a0
Enter up to 8 character source schema ==> nullid
Enter section number (0 for all) ==> 0
Enter outfile name. Default is to terminal ==> /tmp/db2exfmt.out.ok
Output is in /tmp/db2exfmt.out.ok.
Executing Connect Reset -- Connect Reset was Successful.
```

Notice that wildcards, such as %, can be used. The output generated by the explain table formatter is divided into several segments:

1. Explain Instance information
   Includes database context information and package context information.
   - Database context information

Includes the values of the database parameters when the explain table information was obtained. The values provided are only those values that have some impact on the optimizer while determining the access plan.

```
******************** EXPLAIN INSTANCE ********************

DB2_VERSION:            02.01.1

SOURCE_NAME:            SQLC26A0
SOURCE_SCHEMA:          NULLID
EXPLAIN_TIME:           1996-06-28-12.13.47.297612
EXPLAIN_REQUESTER:      DB2A

Database Context:
----------------
        CPU Speed:              0.000031
        Buffer Pool size:       1000
        Sort Heap size:         256
        Lock List size:         100
        Maximum Lock List:      10
        Average Applications:   1
        Locks Available:        1130
```

- Package context information
  Information about blocking and isolation level of the package.

```
Package Context:
---------------
        SQL Type:               Dynamic
        Optimization Level:     5
        Blocking:               Block All Cursors
        Isolation Level:        Cursor Stability
```

2. Statement information
   Provides the original SQL statement and the optimized version of the SQL statement. This section also includes the access plan of the statement and information about tables and indexes involved in the access plan.

```
---------------- STATEMENT 1  SECTION 203 ----------------
        QUERYNO:                1
        QUERYTAG:
        Statement Type:         Select
        Updatable:              No
        Deletable:              No
```

- Original and optimized statement

```
Original Statement:
------------------
select a.name,b.name
from tpcd.region a,tpcd.nation b
where a.regionkey=b.regionkey
order by b.name


Optimized Statement:
-------------------
SELECT Q2.NAME AS "NAME", Q1.NAME AS "NAME"
FROM TPCD.NATION AS Q1, TPCD.REGION AS Q2
```

```
WHERE (Q2.REGIONKEY = Q1.REGIONKEY)
ORDER BY Q1.NAME
```

- Access plan for the optimized statement
  The access plan provides the total cost of the explained statement.

  ```
  Access Plan:
  -----------
          Total Cost:             110.678146
  ```

  Next, it provides information on each step (operator) of the access plan in the following stanzas:

  - The first stanza includes the total cost (in timerons), CPU cost, I/O cost and first row cost of the step. Notice that the operator has been assigned number 10. This number is used to follow the stream of data.

    ```
    10) TBSCAN: (Table Scan)
    Cumulative Total Cost:          53.147976
    Cumulative CPU Cost:            99962.000000
    Cumulative I/O Cost:            2.000000
    Cumulative First Row Cost:      25.510544
    ```

  - Arguments
    Including prefetching, locks and direction of scan.

    ```
    Arguments:
    ---------
    MAXPAGES: (Maximum pages for prefetch) ALL
    PREFETCH: (Type of Prefetch) NONE
    ROWLOCK : (Row Lock intent) SHARE
    SCANDIR : (Scan Direction) FORWARD
    TABLOCK : (Table Lock intent) INTENT SHARE
    ```

  - Input streams
    Identifies the operator or the operand from which it will get its input. In this case, the input is an operand: the TPCD.NATION table.

    ```
    Input Streams:
    -------------
    5) From Object TPCD.NATION

    Estimated number of rows:       25.000000
    Number of columns:              3
    Subquery predicate ID:          Not Applicable

    Column Names:
    ------------
    +$RID$+NAME+REGIONKEY
    ```

  - Output streams
    This identifies the operator that will be the recipient of this output step.

    ```
    Output Streams:
    --------------
    6) To Operator #9

    Estimated number of rows:       25.000000
    Number of columns:              3
    Subquery predicate ID:          Not Applicable
    ```

```
Column Names:
------------
+$RID$+NAME+REGIONKEY
```

- Information about the objects or operands (tables and indexes) used in the access plan.

```
Objects Used in Access Plan:
---------------------------

        Schema: TPCD
        Name:   NATION
        Type:   Table
                        Time of creation:               1996-05-30-11.18....
                        Last statistics update:         1996-06-26-15.06...
                        Number of columns:              4
                        Number of rows:                 25
                        Width of rows:                  39
                        Number of buffer pool pages:    2
                        Distinct row values:            No
                        Tablespace name:                USERSPACE1
                        Tablespace overhead:            24.100000
                        Tablespace transfer rate:       0.900000
                        Prefetch page count:            32
                        Container extent page count:    32
                        Table overflow record count:    0

        Schema: TPCD
        Name:   REGION
        Type:   Table
                        Time of creation:               1996-05-30-11.18....
                        Last statistics update:         1996-06-26-15.05...
```

5.2 Benchmarking Tool

The utility db2batch is a benchmarking tool that provides performance information. This tool processes batch SQL statements. When the db2batch tool is invoked, it will perform the following:

- Connect to the database
- Read, prepare, and execute the SQL statements
- Disconnect from the database
- Return the answer set, allowing you to determine the number of rows to be retrieved and the number of rows to be sent to output
- Return performance information, allowing you to specify the level of detail
- Return the mean values for ‘elapsed time’ and ‘Agent CPU time’ of all the SQL statements executed

The db2batch is usually fed by an input file. In this file, the user is able to set the different options and write the SQL statements that are to be executed by the utility.

```
--#SET perf_detail 2
--#SET rows_fetch 20
--#SET rows_out 10
select name from tpcd.supplier;
--#SET rows_fetch -1
--#SET rows_out -1
select name,regionkey from tpcd.region order by name;
```

When writing an input file, such as the one above, the basic rules are:

1. Options have the syntax `--#SET <option> <value>`. Table 4 lists the different options available.
2. Options apply only to the SQL statements below them.
3. Options can be 'unset' by setting their value to -1.
4. SQL statements must be terminated by a semicolon.

OPTION	VALUE	Comment
perf_detail	0	No performance information
	1	Only elapsed time and average
	2	Agent CPU time, elapsed time and average
	3	Agent CPU, elapsed time, database snapshot and average
	4	Agent CPU time, elapsed time, database manager snapshot, application snapshot, database snapshot and average
rows_fetch	<number>	Rows fetched from answer set
rows_out	<number>	Rows fetched to be sent to output
timestamp		Generate a timestamp
pause		Pause after each SQL statement

Table 4. Options for db2batch

The output of the db2batch utility can be sent to a file. The level of detail, perf_detail, is set to 1 by default. This means that only the elapsed time for each SQL statement, agent CPU time for each SQL statement and the mean value of both will be returned. The default value for rows_fetch and rows_out is -1, meaning to fetch all rows from the answer set and to send all rows fetched to the output device.

Db2batch can be used to get snapshots easily. Setting perf_detail to 4 will get a complete snapshot (database manager, database and application) for every SQL statement included in the input file. Results include the same data elements used by the snapshot monitor.

When benchmarking, setting rows_out to 0 will avoid 'flooding' the output device with the rows fetched.

5.3 Bind File Dump Tool

The db2bfd utility displays the contents of a bind file. It may help to determine how a package was built. The bind file dump utility provides the following information about the bind file:

- The precompile options used to create the bind file and bind file header information as shown in this screen:

```
/usr/lpp/db2_02_01/bnd/db2look.bnd:  Header Contents

.Element name   Description                    Value
-------------  ------------------------------  ------------------
bind_id        Bind file identifier            :BIND V02:
relno          Bind file release number        :0x300:
application    Access package name             :DB2LOOK :
timestamp      Access package timestamp        :XBHmRCCM: 1996/02/02 17:38:07:085
creator        Bind file creator               :NULLID  :
endian         Bit representation              :B: Big Endian (non-Intel)
sqlda_doubled  Indicates if SQLDA doubled      :1:
insert         DB2/PE buffered inserts         :0:
max_sect       Highest section number used     :22:
num_stmt       Number of SQL statements        :156:
statements     Offset of SQL statements        :1364:
declarel       Size of data declarations       :3760:
declare        Offset of data declarations     :32934:
prep_id        Userid that created bindfile    :NULLID  :
date_value     Date/Time format                :0: Default (Default)
stds_value     Standards Compliance Level      :0: SAA (Default)
isol_value     Isolation option                :2: Uncommitted Read (Defined
blck_value     Record blocking option          :1: Block All (Defined)
vrsn_value     Version option                  : : (Default)
...
```

- The SQL statements included in the bind file.

```
/usr/lpp/db2_02_01/bnd/db2look.bnd:  SQL Statements = 156

Line Sec Typ Var Len SQL statement text
---- --- --- --- ------------------------------------------------------------
 147   0  10   0  13 INCLUDE SQLCA
 269   0   5   0  21 BEGIN DECLARE SECTION
 378   0   2   0  19 END DECLARE SECTION
 445   0   5   0  21 BEGIN DECLARE SECTION
 447   0   2   0  19 END DECLARE SECTION
 655   0  19   1  20 CONNECT TO :database
 667   1   0   1 108 SELECT colcount INTO :snbr FROM SYSIBM.SYSTABLES
                           WHERE creator = `SYSIBM' AND name='SYSTABLES'
 771   2  15   6 718 DECLARE C1_SYSTABLES CURSOR FOR              SELECT  T1.t
                     bspace, ntables, name, creator, card, npages,
                                 index_tbspace, long_tbspace, fpages, overflo
                     w             FROM    SYSIBM.SYSTABLES T1,
                                (                            SELECT T2.tbspace as
                      tsname, count(*) as ntables                            FR
                     OM   SYSIBM.SYSTABLES T2                           WHERE
                     T2.TYPE = `T'                              GROUP BY T2.tbspa
                     ce                         ) T3             WHERE   (creat
                     or = :creator_userid OR :creator_userid:ind is NULL)
...
```

5.4 Productivity Tool

The db2look tool helps to create a replica database from an existing one. This replica may be used for benchmarking or tuning purposes. The output of the db2look utility may include:

- The DDL statements to re-create the tables and indexes of the “source” database.
- The SQL statements to update the catalog statistics of the replicated database. Updating the statistics will ensure that the same access plan will be used.
- The alter tablespace commands to set the extent and prefetch sizes of the replicated database to be the same as the “source” database.

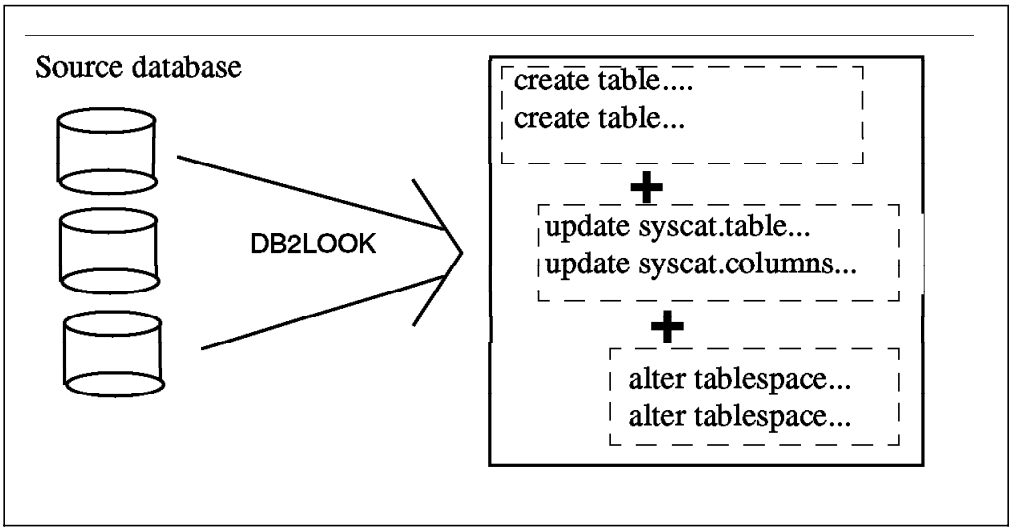


Figure 55. Db2look Utility

Notice that db2look will not build a clone of the “source” database. As it is shown in Figure 55, the output of the db2look utility is a file that will facilitate the creation of a replica database. If trying to clone a database, other steps must by executed by hand:

- Creation of the database, tablespaces and containers
- Creation of triggers and UDFs
- Loading of data into the tables

5.5 DB2 Governor Tool

The db2gov tool acts as a governor of databases. It is designed to limit the use of resources by online DB2 users. It does not prevent access but limits the amount of processing that can be performed by a query.

Db2gov periodically invokes the snapshot monitor, and with the information from the snapshot, it determines what applications are exceeding the limits set in its configuration file. Applications exceeding the limits are terminated, or the priority of the agent servicing the application is changed. Offending applications are posted in a report file. The syntax to invoke the db2gov tool is:

```
db2gov config_file report_file
```

Db2gov can only be used by the database administrator. In order to change the priority of the agent servicing the offending application these two premises must be met:

- For UNIX servers, root has to be the owner of the db2gov program, and the setuid bit must be set.
- The agentpri database manager configuration parameter has to be set to its default value.

If the database manager configuration parameter max_idleagents is not 0. There is a chance that an application will reuse a database agent whose priority has been changed. To avoid this effect, the first line of the configuration file should read:

```
setlimit cpu 1 locks 1 rowsread 1 action priority 0;
```

5.5.1 Configuration File

The configuration file for db2gov includes the following information:

- Name of the database.
- Sleep interval.
  Db2gov invokes the snapshot monitor, looks for applications that are using more resources than they are allowed to and acts on the applications. The process then sleeps for the number of seconds specified in this interval.
- Conditions under which applications are forced or their agents' priorities changed. These conditions can be filtered, and thresholds can be set.
  1. Filters
     - Time of day
     - APPLIDs
     - AUTHIDs
  2. Thresholds
     These thresholds are used to determine if the application should be forced or if its agent's priority should be changed. Limits may be imposed on the following resources:
     - Agent CPU time
     - Number of locks held by the application
     - Number of rows read by the application
     - Elapsed time of the current UOW

The configuration file can be modified while db2gov is being executed. Db2gov will check if the configuration file has been modified during its sleep interval. The format of the configuration file is:

```
interval <seconds>:
dbname <databasename>:
<restriction> setlimit <limits> action <actions>:
<restriction> setlimit <limits> action <actions>:
<restriction> setlimit <limits> action <actions>:
...
```

Restrictions: The restrictions have the following syntax:

```
time <timeclause> authid <authidclause> applname <applnameclause>
```

The format and meaning of the clauses are:

`timeclause` Interval in which the restriction applies. The format is hh:mm hh:mm. The default value is 00:00 23:59.

`authidclause` List of authids, separated by commas, to which the restriction applies. The default is all authids.

`applnameclause` List of application names, separated by commas, to which the restriction applies. The default is all applications.

Limits: The limits have the following syntax:

```
setlimit cpu <nnn> locks <nnn> rowsread <nnn> uowtime <nnn>
```

where

`cpu` Agent CPU time in seconds. If set to -1, unlimited.

`locks` Number of locks held by the application. If -1, unlimited.

`rowsread` Number of rows read by the application. If -1, unlimited.

`uowtime` Number of seconds since the UOW began. If -1, unlimited.

One of the limits must be specified, and no limit can be set to 0. Notice that <nnn> does not represent a 3-digit number but any possible natural number (or -1).

Actions: The action clause specifies the action to take if one of the limits is exceeded. The two possible actions to be taken are:

- Change the priority of the agent. Valid values of priority can range from -20 to 20. The format of the action clause is: `action priority <nnn>`
- Force the application. The format of the action is: `action force`.

If no action is specified, the default is to reduce the priority of the agent by 10.

5.5.1.1 Sample Configuration File

Figure 56 is a sample configuration file for the db2gov utility. Notice that all lines are finished by a semicolon and that comments are enclosed in {}.

```
{db2gov will wake up every 10 seconds}
interval 10;
{our database name is tpcd}
dbname tpcd;
{The next clause is used to avoid problems with priorities of idle agents,
as it will set the priority of any agent to its default value as soon as the
agent has read a 1 row or used 1 second of CPU or is holding one lock}
setlimit cpu 1 locks 1 rowsread 1 action priority 0;

{The next clause will limit the cpu time of any agent to 10 minutes. The
offending application will be forced}
setlimit cpu 600 action force;

{The following clause limits the use of the CLP between 8 AM and 5 PM
to read less than 250 rows}

time 8:00 17:00 applname db2bp_32 setlimit rowsread 250 action force;
```

Figure 56. Sample db2gov Configuration File

5.5.2 Report File

The report file will contain errors posted by db2gov and records of applications forced or that had their priorities changed by db2gov. A report file will look like the file in this example:

```
DB2 Governor operational on Fri Jun 28 16:53:06 1996.
        Config file: db2gov.config
        Report file: db2gov.out
        dbname: tpcd
        Snapshot interval: 10 (seconds).

Fri Jun 28 16:53:06 1996.  Forced application `db2bp_32`, authid `DB2A`.  CPU
seconds = 3.  Locks held = 4.  Rows read = 9126.
Application satisfied restriction on line 5 of configuration file.
```

The application — in this case, the Command Line Processor — is terminated, and db2gov appends its output to the report file, db2gov.out.

5.6 DB2 Simple Network Management Protocol Subagent

The DB2 Simple Network Management Protocol (SNMP) subagent, db2snmp, is based on the RDBMS MIB. The RDBMS MIB contains objects that may be used to manage relational database implementations. Specifically, it contains information on installed databases, on servers, and on the relationship of databases and servers. This MIB requires no writable objects, so it is not expected to be used as the 'active' database management tool.

The RDBMS MIB is formed by three subtrees:

1. RDBMS Objects
   This subtree is made of the following tables:

   rdbmsDbTable
   : This table contains one entry per database installed. It includes the vendor name, database name and database contact.

   rdbmsDbInfoTable
   : This table includes information on the actively opened databases. It includes the product name, its version and the date of the last backup.

   rdbmsDbParamTable
   : This table contains the configuration parameters of the database and their values.

   rdbmsDbLimitedResourceTable
   : This table contains information about limited resources of the database. For DB2 servers, it contains information about connections and the database heap. It includes its current values, high-water marks and configuration limits (maxappls and dbheap).

   rdbmsSrvInfoTable
   : This table maintains information about active database servers. Information is divided into two tables:

   applTable
   : Keeps information about application-specific objects. In this context, the database manager is considered to be a 'network application'. It provides information about the status of the

database manager, name, version, and associations (connections, local and remote).

rdbmsSrvInfoEntry
: Additional information on the database manager: start-up time, finished transactions, disk reads, logical reads.

rdbmsSrvParam
: Contains all the database manager configuration parameters.

rdbmsSrvLimitedResourceTable
: This table maintains information about limited resources of the database servers. It contains information about current agents, high-water marks, and configuration limits (maxagents).

rdbmsRelTable
: The purpose of this table is to keep a track of the relationships between database servers and databases. It contains an entry for each database-database_manager relation.

rdbmsWellKnownLimitedResources
: This tree is for documentation purposes only. It only contains an object for each limited resource (database or database manager) but no values for those objects.

2. rdbmsTraps
   This subtree is made of the following tables:

   rdbmsStateChange
   : An rdbmsStateChange trap signifies that one of the database servers/databases managed by this agent has changed its rdbmsRelState object in a way that makes it less accessible for use. For these purposes, both active and available are considered fully accessible. The possible states are: other, active, available, restricted, unavailable. The state sent with the trap is the new, less-accessible state.

   rdbmsOutOfSpace
   : An rdbmsOutOfSpace trap signifies that one of the database servers managed by this agent has been unable to allocate space for one of the databases managed by this agent. Care should be taken to avoid flooding the network with these traps. It is not implemented in this release.

3. rdbmsConformance. This subtree is made of the following objects:

   rdbmsCompliances
   : Minimum set to be supported by the agent to be compliant with the RFC.

   rdbmsGroups
   : Groups and objects defined in RFC 1697.

The DB2 SNMP subagent will capture snapshot information based on the snapshot monitor switch settings. These settings affect all the databases created under the same instance. By default, it will capture snapshot information every five seconds. This can be configured when the subagent is started. Subagents are started by:

- Executing the following command for AIX servers:

  ```
  db2snmp <instance> {-t nn]
  ```

  The script db2snmp is placed in the /usr/lpp/db2_02_01/cfg directory. The flag -t specifies the interval between snapshots.

- To start the SNMP subagent for OS/2 , you can click on the Start SNMP Subagent icon or execute this command from the command line:

  ```
  db2snmp [-t nn]
  ```

  The subagent will monitor the 'current' instance; `nn` is the interval, in seconds, between snapshots.

- For NT, the subagent is not available.

To stop the SNMP subagent from the command line:

```
db2snmpd -end
```

or click on the Stop SNMP Subagent icon.

Chapter 6. Tasks and Methodology

This chapter covers procedures that may be used when tuning a database environment or when trying to solve performance-related problems. When dealing with performance, the first task is to define the environment and the methodology to be used.

Performance is not an absolute value. The performance of an information system can be measured as better or worse than a reference value. That reference value should be established first, and then the results of tuning efforts can be compared against the reference value.

The reference value has to be established according to the requirements of the information system. Those requirements, or service level agreement, may include the throughput of the system, limits on the response time for a percentile of transactions or any other issue relevant to the end user.

6.1 Defining the Yardstick

Since performance is always a relative attribute, the database administrator will have to define a reference, a unit of measurement for each database. Future performance measurements should be compared to this unit of measurement.

Units of measurement are usually based on the response time of a given workload. Other units of measurement may be based on transactions per second, I/O operations, CPU use or a combination of the above.

6.1.1 Creating a Benchmark

To be able to compare results in different periods of time or under different configurations, a “workload” must be created. This workload should be the same kind of workload as the one created by applications that access the production database.

The workload is a set of queries, SQL statements or database utilities that reproduce the same type of work generated by the applications that will be accessing the data contained in the database environment. The workload must have the following two properties:

- The workload should not destroy the environment when executed.
- The workload must be reusable.

If the applications are static SQL applications, the workload will be imbedded in an application. It may also be desirable to include some measuring tools or method within the same application. To accomplish this task, DB2 provides extensive APIs to all the data elements that monitor the instance, the database or the application.

For dynamic SQL applications, there are different possibilities. The workload can also be included in a program, like with the static SQL. Another alternative is to use the db2batch utility, which is discussed in 5.2, “Benchmarking Tool” on page 104. This utility provides a simple way to capture the data elements through snapshots. Db2batch also computes response time of the SQL statements.

6.1.2 Environment

When dealing with performance issues for a database environment, it is not always possible to fully test modifications to configuration parameters or applications because, in many cases, you may be dealing with a production environment. Testing against a production database is not possible when:

- Changes to configuration parameters require you to stop and restart the instance or require all applications to disconnect from the production database.
- The tests include modifications (updates, deletes, inserts) of data.
- The tests include changes in the way data is accessed, such as different data placement, different runstats or changes in the optimization level or access plans.
- Testing requires recompiling and rebinding of applications.

A common approach is to have a trial or development instance with copies of production databases. In this environment, configuration parameters can be easily modified. This “trial” instance solves some of the problems found when testing against production databases. But having this testing environment raises other issues:

- Resources may not be available (machine, disk, memory...).
- The environment is not exactly the same as the production environment.
- Some tests may require you to simulate the workload of other users.
- Some tests may require you to simulate concurrent users.
- The volume of data of the trial databases may not be the same.
- The contents or placement of data of the trial databases may not be the same as those of the production databases.
- Data may be accessed through different access plans than those of the production databases.
- Any modification derived from this environment will have to be implemented in the production environment.

So a balance must be achieved between testing against the production databases and testing against trial databases. One one hand, you get the real environment; on the other hand, you have the freedom and availability to test modifications.

Some tests, such as those designed to monitor performance, can be executed against production databases. The basic rule when testing against production databases is that data cannot be modified. This is possible if tests are query-only or if they roll back any executed transaction. Other tests, those involving changes in configuration parameters or data, will have to be executed against trial databases.

DB2 is shipped with a tool called db2look that will help the database administrator create trial databases. This tool can create a file with the DDL statements of an existing database and can also create the SQL statements that allow you to clone the statistics of the existing/production database. This file can then be used as the input for the Command Line Processor to create a similar database environment to the existing production database.

After a trial database is created, data has to be inserted into the tables. Often, it is not possible to insert the same volume of data due to space restrictions. A subset of data has to be selected and placed into the trial database. Working with subsets of data limits the viability of some performance measurements.

6.1.3 Measuring and Monitoring

The results obtained when testing or benchmarking can be split into two different categories:

- Attributes Being Measured

  Usually, the response time is the only measured attribute. But any other metric, such as a throughput, may be the target.

- Attributes Being Monitored

  These attributes show the behavior of the database during the test. For example, you may be interested in the way pages are being accessed through prefetchers, or the watermark of a heap left by the execution of a test. DB2 provides a variety of tools for this purpose: the event monitor tool, the snapshot monitor tool and the performance monitor tool.

When measuring and monitoring, each of the steps of the test should be carefully considered. The method you are going to use to measure must be clearly established. You should be aware that the measuring and monitoring activity may have some affect on the results obtained. Previous steps of the test, or previous tests, may also have an impact because pools and caches are not flushed unless you take steps to guarantee a clean measure.

Reports of the results should be kept for future performance references. Performance is a relative issue, and progress is best tracked based on previous performance issues. It makes sense to define what information should be kept in a performance report. Reports should include not only the results, but also the configuration parameters, the environment and values of other monitored attributes.

6.2 Monitoring Performance

Periodic measurements on production databases can help provide the answer to the question: How is my production database performing? These periodic measures may be taken using any of the tools provided by DB2. An example of the measures that can be taken using the available tools are:

- Creating an event monitor for each production database.

  This event monitor will provide information of a “working-day”, as information of the data elements is written when the last application disconnects from the database. Analysis of high-water marks can be compared to configuration parameter values. Overflows and rejections point to improper configuration values.

- Creating event monitors for connections to active databases.

  This event monitor will record an event each time an application connects to and disconnects from the database. The distribution of connections can be obtained from this monitor, as well as conclusions on the size of configuration parameters for the number of agents and concurrent agents. Obtaining the distribution of connects/disconnects may help the database

administrator to schedule maintenance periods or to set the time frame for the database backup policy.

- Creating event monitors for tables.

  Event monitors for tables are useful to determine which tables are more-frequently accessed. Such monitors may suggest changes in the physical design of the database.

- Getting a snapshot for each production database at peak times.

  Counters can be reset before taking the snapshot, for example, one hour before. This will give cumulative values for the data elements during the peak hour. Compare high-water marks to values of configuration parameters. Check for overflows, rejections, etc.

- Using the performance monitor.

  Resetting counters and establishing a one-hour interval between snapshots will provide the same information as getting a snapshot on a peak hour. If the interval is an eight-hour interval, you will obtain information of a "working-day" such as that information provided by an event monitor for a database.

- Obtaining response times.

  The response time of the workload defined in 6.1.1, "Creating a Benchmark" on page 113 can be measured. These values should be compared with previous measurements.

The information collected can be compared on a daily or weekly basis to detect trends on data elements. Analyzing trends may indicate corrective actions that can be taken even before any performance problems appear.

Special attention may be taken after important changes are made, such as the loading of large amounts of data or the modification of an important database configuration parameter.

6.3 Tuning for Performance

One question in the mind of database administrators is: How can my database perform better? Unrelated to a specific performance problem, the goal here is to reduce the response time of applications or to minimize the resources allocated to the database while maintaining the same level of response time.

The default configuration values in DB2 are oriented to machines with relatively small amounts of memory and which are dedicated as database servers. Configuration parameters should be modified when using large databases, unique query or transaction loads or a large number of users.

It should also be kept in mind that tuning a database may not solve complex performance problems. Through tuning, an important benefit can be achieved. But it will not eliminate problems created by poor physical design of a database or its applications.

When looking at tuning a database the following points are a guideline for the tasks that need to be accomplished:

- Create or define the testing environment.
- Define the measuring unit.

- Create or define the workload to be measured.
- Submit the workload, and measure the results.
- Modify only one configuration parameter.
- Reestablish the original testing environment.
- Submit the workload, and measure the results.
- Compare both sets of results obtained.

Choosing which of the configuration parameters to modify, and the decision to increase the parameter or decrease it, will depend upon the objectives set by the database administrator. If the objective is to reduce the response time of applications, the database administrator will have to identify where the database manager currently spends its time and resources when obtaining the answer sets requested by applications. For example, in general, the response time can be lowered by:

- Reducing the number of I/O operations
- Avoiding 'waits'
- Cutting down the CPU requirements of applications
- Sorting and joining properly
- Returning the answer set to the application in big blocks of data

6.3.1 I/O Operations

Performance can be increased by reading more pages in each I/O operation or by reducing the number of I/O operations required to obtain the answer set. The number of I/O operations will be affected by both the settings of configuration parameters and by the database design or operation.

Database Configuration: The number of pages read in each read operation will depend upon whether prefetchers are being used. Deciding on whether to prefetch or not is determined by the database manager, as the database administrator can only turn off sequential prefetching, through the seqdetect configuration parameter), but cannot force prefetching. The number of pages read when prefetching is performed is limited by the prefetchsize configuration parameter of each tablespace or by the def_prefetch_sz database configuration parameter.

Prefetcher activity is monitored through its related data elements (buffer pool data or index reads and buffer pool asynchronous reads). Tools that display the values of these data elements are the performance monitor, the snapshot monitor (for the database or for a tablespace) and the event monitor (database or tablespace also). The performance monitor not only shows the values of these data elements but also calculates buffer pool hit ratios and can graphically display results over a period of time.

The number of I/O operations can be reduced by proper configuration of caches and pools so that pages already resident in memory areas remain there for subsequent requests. Data pages read stay in the buffer pool, table descriptors reside in the catalog cache, and package-related information resides in the package cache. These three memory areas are configured through the buffpage, catalogcache_sz and pckcachesz database configuration parameters.

A hit ratio for these three memory areas is provided by the performance monitor, as well as values for their related data elements. Values for these data

elements are also provided through a snapshot monitor for the database or through an event monitor for the database.

Database Design: Indexes may be chosen by the optimizer to process queries. Access to data through indexes may improve the response time by a huge factor. It is possible to use the `explain` utility to see if data is being accessed through indexes. Also, by keeping statistics updated, using the `runstats` utility will allow the optimizer to choose the best access plan for applications.

Clustering data by the most accessed index can greatly reduce the number of I/O operations required to retrieve answer sets. With this in mind, you should consider reorganizing tables frequently if the data is dynamic.

Using the `reorgchk` utility, you are able to determine when to reorganize tables. If you want to reorganize a table based on an index, the `reorg` utility can be used.

Pages can be read in parallel from different physical drives. This will depend on the physical design of the database. Read operations can be executed in parallel if the following conditions are met:

- Prefetching is being performed.
- The tablespace containing the table being accessed is distributed through several containers.
- Containers are placed in different disk drives.
- The number of I/O servers used by the database is more than one. The number of I/O servers is configured through the `num_ioservers` parameter.

By default, write operations to log files will be performed each time a transaction is committed. Write operations to log files can be reduced by grouping commits together. This will reduce the number of writes, thus improving the performance of the database. But it will penalize the response time of small transactions, as they will have to wait for other transactions to commit their work. If there are not enough transactions ready to commit their work, the database manager will commit transactions every second. The number of transactions that can be grouped together is set by the `mincommit` configuration parameter.

The number of write operations is also higher if there is not enough space in the log buffer. If a unit of work fills the log buffer, data will be written to the log files. In order to reduce the number of unnecessary writes, the `logbufsz` configuration parameter should be set to fit the log space requirements for the different units of work being performed on the databases.

The amount of log space required by a unit of work can be monitored using the performance monitor or the snapshot monitor for applications, or by creating an event for the transaction that is to be measured.

I/O cleaners are triggered when the number of free pages in the buffer pool exceeds their defined threshold. When triggered, they build a list of modified pages and then write these pages to disk. The threshold value is set by the `maxchngpgs` configuration parameter. The number of processes that will perform this task is set by the `num_iocleaners` configuration parameter. To minimize I/O activity, you can increase the size of the buffer pool or increase the threshold that triggers the I/O cleaners. To write pages faster, set a proper number of cleaners. As a guideline, you should have as many page cleaners as there are

storage devices. Notice that setting maxchngpgs to a very high value may force applications to wait for free pages in the buffer pool, and so you should monitor the database carefully after any such changes.

There are data elements that monitor the number of times I/O cleaners are triggered. For example, you can monitor elements such as the buffer pool threshold cleaners triggered, which counts the number of times that I/O cleaners have been triggered because the threshold of “dirty pages” has been reached. This data element is provided by the performance monitor and by the snapshot monitor tools.

6.3.2 Waits

A way to reduce the response time of applications is to avoid having applications wait. Applications will wait for the following:

- I/O operations.

  Applications will wait for synchronous I/O operations. Many synchronous reads can be substituted by asynchronous reads performed by prefetchers.

  Synchronous and asynchronous reads can be monitored though the performance monitor, the snapshot monitor or the event monitor.

- Free pages in the buffer pool.

  If there are no free pages in the buffer pool, the agent serving the application will have to “clean” pages on its own. I/O cleaners assure that agents will find free pages in the buffer pool.

  There is not a data element that will keep track of the free pages in the buffer pool. An undersized buffer pool is easily detected by the number of times I/O cleaners are triggered. This number can be monitored through the performance, snapshot or event monitors.

- Locks.

  The agent serving an application will have to wait if the tables or rows required are locked by other agents. Waiting on locks depends on the level of isolation used by applications. Waiting time can be severely affected by lock escalations. To avoid lock escalation, set proper values to the locklist (memory area for locks) and maxlocks (max percentage of the lock list that can be held by an agent) configuration parameters. Notice that lock waits or lock escalations can be minimized, but usually cannot be entirely eliminated in concurrent systems that provide data integrity.

  Again, the performance, snapshot or event monitors can be used to monitor the locking that occurs within the database or application.

- Prepare and bind.

  Dynamic SQL statements have to be precompiled and bound unless they are already found in the package cache of the agent from a previous execution. The time expended in the prepare and bind is heavily influenced by the level of optimization used. The objective is to find the balance between the level of optimization and the execution time. Query optimization levels for a package can be displayed using the db2bfd, db2exfmt, db2expln or Visual eExplain utilities.

- An agent.

  When an application connects to a database, the database manager will create an agent (a process or a thread) to attend to its database requests.

To avoid the having to wait until the agent is created, a number of idle agents may exist. The number of idle agents that a database may have is determined by the max_idleagents configuration parameter.

- For the first connection.

  When the first application connects to a database, it will have to wait for the database global memory area to be allocated. This can be avoided through the use of the command db2 activate database.

6.3.3 CPU Requirements

The performance of applications will improve if the number of CPU cycles required to retrieve the answer set is diminished. This can be achieved by using the optimizer. The optimizer can rewrite queries and determine the access plan for the most cost-effective execution of SQL statements. The amount of time spent by the optimizer and the type of optimization algorithm is determined by the optimization level. Notice that the higher the level of optimization, the better the execution time that should be achieved. But also, higher optimization levels imply that the optimizer will take more time to perform its tasks, which can impact the performance of dynamic SQL statements.

The access plan is not only influenced by the level of optimization. The optimizer uses the catalog statistics when choosing an access plan. Statistics should be updated, via the runstats utility, when the amount of data stored in the tables changes considerably.

Isolation levels and query optimization levels for a package can be displayed using several of the tools available.

6.3.4 Sorts and Joins

Sorts and joins may have a big impact on application performance. When creating indexes, you should consider defining them on the columns that will be used in the join predicate. Often, sorting can be avoided, if well-clustered indexes are used. However, even if indexes exist, a sort may occur if the optimizer determines that it is less expensive than using indexes to retrieve the answer set. The explain facilities (Visual Explain, db2expln) show the number of sorts required to execute an SQL statement.

When dealing with sorts, piped sorts have a performance advantage. Piped sorts rejections can be monitored, as well as sort overflows or post threshold sorts. These elements monitor the size of the sort heap and the limit imposed by the sort threshold. Performance and snapshot monitors can be used to display the value of these elements. Sort overflows can also be shown by creating an event monitor for the statement.

6.3.5 Block I/O

When an application retrieves a large answer set during a read-only operation, application performance will improve if data is sent from the server to the client in big blocks. In non-blocking operations, for each fetch request from the client, one row is returned by the server. When blocking is used, a set of rows will be returned by the server for each client fetch request.

The size of the memory areas used for blocking cursors is configured through the aplheapsz (for local clients) and the rqrioblk (for remote clients) configuration parameters.

6.4 Solving Performance-Related Problems

When an application is expected to have a better response time than is being obtained, the process of diagnosing the problem starts. If the workload on a system increases, the throughput of the system may be maintained, but individual application response time may be degraded.

6.4.1 Describing the Cause

As in any other problem-determination technique, the first step requires the database administrator to be able to reproduce or identify the problem. Some symptoms may be sporadic; others may be permanent.

After the problem is reproduced or identified, it will fall into one of these two groups:

1. Problems affecting all applications

   Problems that affect all applications usually appear when changes are made to data loads, the number of users, the operating system, or the database configuration parameters. The cause of these types of problems is usually found in the following areas:

   - Configuration parameters (sorts, buffer pool, logs, lock list). Sometimes, this is not caused by a modification in the parameter but by the environment itself. Bigger tables that require a larger sort heap or the updating of more rows in a table may require a larger log buffer. Also, more users exhausting the lock list and provoking concurrence problems can cause problems that affect all database applications.
   - Operating system problems, such as I/O contention or excessive paging.
   - Network problems, if the clients or applications are remote.
   - Data access. Access plans may be obsolete, statistics may not have been updated or packages may not be rebound.

2. Problems affecting one application or a group of applications

   Problems that affect a single application or a group of applications can be furtherly subdivided into two categories:

   - Applications that have had a good performance history in a development/testing environment but do not perform as expected working against the production database/databases. Working against low volumes of data may hide problems. Some of the non-detected problems may be those associated with casting, lack of indexes, joins, sorts, access plans, isolation levels or size of the answer set.
   - Applications whose behavior is erratic. These applications may usually have good response times. But under certain conditions, their response times are very degraded. These applications may have concurrence-related problems: deadlocks, waits, etc.

6.4.2 Database Configuration Problems

Many of the database configuration problems are detected through SQLCODEs or SQLSTATEs that are returned to applications. If the error handler routines are well written, they should present the error to the end user. These errors should lead to the cause of the problem and will facilitate the problem-determination process.

When a database configuration problem is suspected, but there is no certainty about the conflicting parameter or parameters, the database should be monitored. This monitoring can be achieved using the performance monitor, the snapshot monitor, the event monitor or a combination of these. When planning to monitor the database environment, you need to choose a significative period of time for monitoring to take place; such as a 60-minute interval during peak hours. Take time to examine the output collected from the monitoring tools, and check for data elements that can point to specific configuration problems. These data elements can be high-water marks, overflows or rejections.

Define a method for resolving problems, and stick to it. The following points can be used a guidelines when establishing a problem-determination method:

- Choose the monitoring tool.
- Define the period of time and environment in which the database will be monitored.
- Start monitoring the database
- Obtain the results of the monitoring tool.
- Based on the results of the monitored data elements, select the parameter to be modified. Modify only the selected configuration parameter. Remember to restart the database.
- Reestablish the original monitoring environment, if possible.
- Monitor again, obtaining the results of the data elements with the new value of the configuration parameter.
- Compare the results obtained.
- If results are not positive, reestablish the configuration parameter to its old value.

When a database is monitored, a large set of values for data elements may be collected. Some values may point directly to the cause of a problem, but that is not always the case. Table 5 on page 123 shows some of the data elements that can be collected, their related configuration parameters and the problems caused by incorrect configuration values. The table is only an example and is not intended as a complete listing for problem determination. Many of the data elements collected will relate directly to configuration parameters. When using the performance monitor, is it possible through the online help to identify which configuration parameter relates to the data elements.

Data Element	*Configuration Parameters*	*Probable cause*
Catalog cache full	catalogcache_sz	Catalog cache too small
Catalog cache heap full	catalogcache_sz dbheap	dbheap too small compared to catalog Cache. dbheap may fill up if the size of the buffer pool is increased.
Post threshold sorts	sortheap sheapthres	Sort heap threshold too small
Sort overflows	sortheap	If too many, applications are requiring a bigger sort heap
Lock timeouts	locklist	If too many, concurrence problem
Lock waits	locklist	If too many, concurrence problem
Deadlocks		If too many, concurrence problem
Lock escalations	locklist maxlocks	Lock list too small, applications monopolizing the lock list.
Buffer pool threshold Ccleaners triggered	buffpage maxchngpgs	If too many, the buffer pool is running out of free pages too often, buffpage too small, or maxchngpgs too low.
Buffer pool log space cleaners triggered	logfilsiz logprimary softmax	If too many, the database is writing to log files too often (to reduce crash recovery time), insufficient sizing of log files or softmax too low

Table 5. Data Elements and Configuration Problems

The best way to evaluate if there are “too many” overflows, timeouts, or waits is to compare results to previous results or a similar environment. When possible, results should be compared to those obtained when the database did not have a performance problem.

Concurrence problems occur only when more than one application is accessing the database. They may point to an application problem related to the isolation levels being used. They also may point to an insufficient size of the lock list memory area. Notice that a snapshot monitor for locks may be taken. This can provide valuable information to determine the cause of problem.

Data elements can be grouped to obtain ratios. Ratios are presented by the performance monitor or can be calculated by the database administrator. The *DB2 Database System Monitor Guide and Reference - for common servers* (S20H-4871) contains a description of all the data elements for which information can be collected and how to calculate ratios using the different elements. An example of these ratios and their relationship to configuration parameters is shown in Table 6.

Ratio	**Configuration Parameters**	**Probable Cause**
Buffer pool hit ratio	buffpage	If too low, prefetchers not working, buffer pool too small
Buffer pool index hit ratio	buffpage	If too low, prefetchers not working, buffer pool too small
Catalog cache hit ratio	catalogcache_sz	If too low, catalog cache too small

Ratio	Configuration Parameters	Probable Cause
Percentage of rollbacks due to deadlocks		If significant, and the number of rollbacks is high, concurrence problem
Percentage of sorts that overflows	sortheap	If significant, sort heap too small

Table 6. Ratios and Configuration Problems

6.4.3 Data Access Problems

Data access is the most probable cause of performance problems that can affect all applications. To avoid these problems, the following steps can be taken:

- The database administrator should reorganize tables periodically.
- The database administrator should keep the statistics updated and should periodically bind/rebind applications.

To check if reorganizations are required, DB2 provides the reorgchk utility. When a performance problem is suspected, and data access is suspected to be the cause, the database tables should be checked. If reorganizations show a low clustering ratio for the clustering index of a table, then that table should be reorganized. If clustered indexes are not being used, a big performance gain could be obtained by reorganizing tables according to the most accessed index.

Operating system tools indicating I/O contention may point to problems with the physical design of the database, such as placement of containers across physical drives, or containers allocated to tablespaces.

6.4.4 Application Problems

There are two basic procedures to use when determining an application problem. These are:

- Explaining the statement

  This can be done through Visual Explain or any of the explain tools. The access plan will show the work that the database manager needs to perform to retrieve the answer set. This should be compared to the access plan expected by the database administrator.

  The access plan provides information not only about the work that the database manager has to perform, but how the work will be done. It will clarify if data is being accessed through indexes or not, and it will provide information about the order of all other operators/operations (index scans, table scans, sorts, merge joins, nested loop joins, inserts, temporary table creations, fetches).

  If not satisfied with the access plan being shown, the database administrator may obtain different access plans for different levels of optimization without needing to execute the statement.

  For dynamic SQL statements, different levels of optimization will deliver different access plans, but they also will show different times needed to prepare the statement. The balance of the time required to prepare the statement and the time required to execute it will yield to a better performance of the statement.

- Monitor the application/database

  The application can be monitored through the performance monitor, taking a snapshot of the application or defining an event monitor for the statements or the transaction. For dynamic SQL statements, the db2batch tool also will provide snapshots (for the application, database and instance, if the appropriate level of detail is selected) and will measure the response time of the statement/statements. Application monitoring will collect values for data elements that can point to performance problems.

  Data elements whose values are collected when monitoring the application include: deadlocks; lock escalations; lock waits and lock wait time; index and data reads and writes; number of sorts and sort time; and the package cache hit ratio.

Appendix A. DATABASE 2 Sizing Worksheets

This appendix includes some guideline information designed to help you size your disk and memory requirements. More detailed information on the sizing of your environment can be found in the planning guides for the platforms you are using.

A.1 DB2 for AIX

The following tables are recommended guidelines for both disk and memory. These may be used as a starting point to help you determine your overall sizing requirements for clients or servers on the AIX platform.

Function	Memory (MB)[a]	Disk (MB)[f]
DB2 for AIX (base)[b]	3.20	26.40
For additional local concurrent users or applications	0.30	-
For each additional concurrent database[c]	1.80	14.00
For each additional concurrent instance	0.45	0.80
Database Director	-	34.00
Documentation - IPF format[g]	-	10.00
Documentation - PostScript format	-	20.00

Table 7. Sizing DB2 for AIX Single-User

Function	Memory (MB)[a]	Disk (MB)[f]
DB2 for AIX Server (base)[d]	3.10	26.9
For additional local concurrent users or applications	0.30	-
For each additional remote client	0.25	-
For each additional concurrent database[c]	1.80	14.0
For each additional concurrent instance	0.45	0.8
Database Director	-	34.0
Documentation - IPF format[g]	-	10.00
Documentation - PostScript format	-	20.00
For each additional client connecting from DB2 for MVS/ESA, DB2 for VSE and VM, or DB2 for OS/400	0.30	-

Table 8. Sizing DB2 for AIX Server

Function[e]	Memory (MB)[a]	Disk (MB)[f]
DB2 Client Application Enabler for AIX (base)	0.15	9.2
For each additional client application	0.15	-
Documentation - IPF format	-	10.0
Documentation - PostScript format	-	20.0

Table 9. Sizing DATABASE 2 Client Application Enabler for AIX (Remote Client)

Function[e]	Memory (MB)[a]	Disk (MB)[f]
DB2 SDK for AIX (base)	0.15	13.8
For each additional client application	0.15	-
Database Director	-	34.0
Documentation - IPF format[g]	-	10.0
Documentation - PostScript format	-	20.0

Table 10. Sizing DATABASE 2 Software Developer's Kit for AIX

Function	Memory (MB)[a]	Disk (MB)[f]
DDCS for AIX (base)	0.10	25.6
For each active connection from remote clients	0.30	-
For each active connection from local clients	0.30	-
Database Director	-	21.5
Documentation - IPF format[g]	-	10.0
Documentation - PostScript Format	-	20.0

Table 11. Sizing DDCS for AIX Multi-User Gateway

A.2 DB2 for OS/2

The following tables are recommended guidelines for both disk and memory. These may be used as a starting point to help you determine your overall sizing requirements for clients or servers on the OS/2 platform.

Function	Memory (MB)[a]	Disk (MB)[f]
DB2 for OS/2(base)[b]	2.9	19
For additional local concurrent users or applications	0.11	-
For each additional concurrent database[c]	1.6	5.5
For each additional concurrent instance	0.2	-
Database Director	-	11.5
Documentation	-	6.2

Table 12. Sizing DB2 for OS/2 Single-User

Function	Memory (MB)[a]	Disk (MB)[f]
DB2 for OS/2 Server (base)[d]	2.9	17
For additional local concurrent users or applications	0.11	-
For each additional remote client	0.1	-
For each additional concurrent database[c]	1.6	5.5
For each additional concurrent instance using NetBIOS	1.0	0.8
For each additional concurrent instance using any other protocol	0.3	0.8
Database Director	-	11.5
Windows Support	-	5.6
Documentation	-	4.0
For each additional client connecting from DB2 for MVS/ESA, DB2 for VSE and VM, or DB2 for OS/400	0.4	-

Table 13. Sizing DB2 for OS/2 Server

Function[e]	Memory (MB)[a]	Disk (MB)[f]
DB2 Client Application Enabler for OS/2(base)	0.7	7.0
For each additional client application	0.3	-
Documentation	-	0.5

Table 14. Sizing DATABASE 2 Client Application Enabler for OS/2 (Remote Client)

Function[e]	Memory (MB)[a]	Disk (MB)[f]
DB2 SDK for OS/2(base)	0.7	10.0
For each additional client application	0.3	-
Database Director	-	11.5
Documentation	-	5.5

Table 15. Sizing DATABASE 2 Software Developer's Kit for OS/2

Function	Memory (MB)[a]	Disk (MB)[f]
DDCS for OS/2(base)	0.7	8
Database Director	-	5.5
Windows Support	-	5.6
Documentation	-	2.7

Table 16. Sizing DDCS for OS/2 Single-User Gateway

Function	Memory (MB)[a]	Disk (MB)[f]
DDCS for OS/2 (base)	0.8	8.2
For each active connection from remote clients	0.30	-
For each active connection from local clients	0.7	-
Database Director	-	5.5
Windows Support	-	5.6
Documentation	-	2.7

Table 17. Sizing DDCS for OS/2 Multi-User Gateway

A.3 DB2 for NT

The following tables are recommended guidelines for both disk and memory. These may be used as a starting point to help you determine your overall sizing requirements for clients or servers on the Windows NT platform.

Function	Memory (MB)[a]	Disk (MB)
DB2 for Windows NT Single-User (base) [b]	8.0	18.0
For first local application	3.0	-
For additional concurrent user or application	0.5	-
For each additional concurrent database[c]	2.0	6.5
For each additional concurrent instance	8.0	-
Documentation	-	6.0

Table 18. Sizing DB2 for Windows NT Single-User

Function	Memory (MB)[a]	Disk (MB)
DB2 for Windows NT Server (base)[d]	8.0	16.0
For first local connection	3.0	-
For each additional local connection	0.5	-
For first remote connection	6.5	-
For each additional remote client	0.5	-
For each additional concurrent database[c]	2.0	6.0
For each additional concurrent instance	8.0	-
Documentation	-	4.0

Table 19. Sizing DB2 for Windows NT Server

Function[e]	Memory (MB)[a]	Disk (MB)
DB2 Client Application Enabler for Windows NT (base)	-	8.0
For each additional client connection	2.0	-
Documentation	-	1.0

Table 20. Sizing DATABASE 2 Client Application Enabler for Windows NT

Function[e]	Memory (MB)[a]	Disk (MB)
DB2 SDK for Windows NT (base)	-	11.0
For each additional client connection	2.0	-
Documentation	-	6.0

Table 21. Sizing DATABASE 2 Software Developer's Kit for Windows NT

Function	Memory (MB)[a]	Disk (MB)
DDCS for Windows NT (base)	-	9.5
For first connection to an application[h]	7.0	-
Documentation	-	3.0

Table 22. DDCS For Windows NT Single-User

Function	Memory (MB)[a]	Disk (MB)
DDCS for Windows NT (base)	3.0	10.0
For first connection from a remote client	2.0	-
For each additional connection from remote clients	0.5	-
For each active connection from local clients	2.0	-
Documentation	-	3.0

Table 23. DDCS For Windows NT Multi-User Gateway

Table Notes:

a Additional memory for some database configuration parameters may be required.

b Assumes one connection to local database, on one instance.

c This is for default database definition. Additional space may be required for table definitions or additional log files.

d Assumes one remote connection to local database, on one instance.

e For memory and disk requirements for other platforms, refer to the documentation supplied with the platform-specific product.

f Additional temporary disk space is required during installation. Refer to the platform's planning guide for details.

g This is the approximate size for English manuals. Other languages may vary.

h This is in addition to the application memory requirement.

Appendix B. Special Notices

This publication is intended to help the system or database administrator to perform detailed database performance analysis and understand how to tune the database environment to gain optimal performance. The information in this publication is not intended as the specification of any programming interfaces that are provided by DB2 for Common Server. See the PUBLICATIONS section of the IBM Programming Announcement for DB2 for Common Server Version 2.1.1 for more information about what publications are considered to be product documentation.

References in this publication to IBM products, programs or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM′s product, program, or service may be used. Any functionally equivalent program that does not infringe any of IBM′s intellectual property rights may be used instead of the IBM product, program or service.

Information in this book was developed in conjunction with use of the equipment specified, and is limited in application to those specific hardware and software products and levels.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, 500 Columbus Avenue, Thornwood, NY 10594 USA.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact IBM Corporation, Dept. 600A, Mail Drop 1329, Somers, NY 10589 USA.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The information contained in this document has not been submitted to any formal IBM test and is distributed AS IS. The information about non-IBM (″vendor″) products in this manual has been supplied by the vendor and IBM assumes no responsibility for its accuracy or completeness. The use of this information or the implementation of any of these techniques is a customer responsibility and depends on the customer′s ability to evaluate and integrate them into the customer′s operational environment. While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

The following document contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples contain the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

Reference to PTF numbers that have not been released through the normal distribution process does not imply general availability. The purpose of including these reference numbers is to alert IBM customers to specific information relative to the implementation of the PTF when it becomes available to each customer according to the normal IBM PTF distribution process.

The following terms are trademarks of the International Business Machines Corporation in the United States and/or other countries:

AIX	DATABASE 2
DB2	DRDA
IBM	OS/2

The following terms are trademarks of other companies:

C-bus is a trademark of Corollary, Inc.

Java and HotJava are trademarks of Sun Microsystems, Inc.

PC Direct is a trademark of Ziff Communications Company and is used by IBM Corporation under license.

UNIX is a registered trademark in the United States and other countries licensed exclusively through X/Open Company Limited.

Microsoft, Windows, and the Windows 95 logo are trademarks or registered trademarks of Microsoft Corporation.

C + +	American Telephone and Telegraph Company, Inc.
HP/UX	Hewlett-Packard Company
Intel	Intel Corporation
Notes	Lotus Development Corporation
Solaris	Sun Microsystems, Inc.
X/Open	X/Open Company Limited

Other trademarks are trademarks of their respective companies.

Appendix C. Related Publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

C.1 International Technical Support Organization Publications

For information on ordering these ITSO publications see Appendix D, “How To Get ITSO Redbooks” on page 137.

- *DB2 Version 2 Planning Guide for Database Administrators*, SG24-2523

A complete list of International Technical Support Organization publications, known as redbooks, with a brief description of each, may be found in:

International Technical Support Organization Bibliography of Redbooks, GG24-3070.

C.2 Redbooks on CD-ROMs

Redbooks are also available on CD-ROMs. **Order a subscription** and receive updates 2-4 times a year at significant savings.

CD-ROM Title	**Subscription Number**	**Collection Kit Number**
System/390 Redbooks Collection	SBOF-7201	SK2T-2177
Networking and Systems Management Redbooks Collection	SBOF-7370	SK2T-6022
Transaction Processing and Data Management Redbook	SBOF-7240	SK2T-8038
AS/400 Redbooks Collection	SBOF-7270	SK2T-2849
RISC System/6000 Redbooks Collection (HTML, BkMgr)	SBOF-7230	SK2T-8040
RISC System/6000 Redbooks Collection (PostScript)	SBOF-7205	SK2T-8041
Application Development Redbooks Collection	SBOF-7290	SK2T-8037
Personal Systems Redbooks Collection	SBOF-7250	SK2T-8042

C.3 Other Publications

These publications are also relevent as further information sources.

- *DB2 Information and Concepts Guide - for common servers Version 2, S20H-4664*
- *DB2 Administration Guide - for common servers Version 2, S20H-4580*
- *DB2 Database System Monitor Guide and Reference - for common servers Version 2, S20H-4871*
- *DB2 Command Reference - for common servers Version 2, S20H-4645*
- *DB2 SQL Reference - for common servers Version 2, S20H-4665*
- *DB2 Application Programming Guide - for common servers Version 2, S20H-4643*
- *DB2 Messages Reference - for common servers Version 2, S20H-4808*
- *DB2 Problem Determination Guide - for common servers Version 2, S20H-4779*

Appendix D. How To Get ITSO Redbooks

This section explains how both customers and IBM employees can find out about ITSO redbooks, CD-ROMs, workshops, and residencies. A form for ordering books and CD-ROMs is also provided.

This information was current at the time of publication, but is continually subject to change. The latest information may be found at URL http://www.redbooks.ibm.com.

D.1 How IBM Employees Can Get ITSO Redbooks

Employees may request ITSO deliverables (redbooks, BookManager BOOKs, and CD-ROMs) and information about redbooks, workshops, and residencies in the following ways:

- **PUBORDER** — to order hardcopies in United States
- **GOPHER link to the Internet** - type GOPHER.WTSCPOK.ITSO.IBM.COM
- **Tools disks**

  To get LIST3820s of redbooks, type one of the following commands:

  ```
  TOOLS SENDTO EHONE4 TOOLS2 REDPRINT GET SG24xxxx PACKAGE
  TOOLS SENDTO CANVM2 TOOLS REDPRINT GET SG24xxxx PACKAGE (Canadian users only)
  ```

  To get lists of redbooks:

  ```
  TOOLS SENDTO USDIST MKTTOOLS MKTTOOLS GET ITSOCAT TXT
  TOOLS SENDTO USDIST MKTTOOLS MKTTOOLS GET LISTSERV PACKAGE
  ```

  To register for information on workshops, residencies, and redbooks:

  ```
  TOOLS SENDTO WTSCPOK TOOLS ZDISK GET ITSOREGI 1996
  ```

  For a list of product area specialists in the ITSO:

  ```
  TOOLS SENDTO WTSCPOK TOOLS ZDISK GET ORGCARD PACKAGE
  ```

- **Redbooks Home Page on the World Wide Web**

  http://w3.itso.ibm.com/redbooks

- **IBM Direct Publications Catalog on the World Wide Web**

  http://www.elink.ibmlink.ibm.com/pbl/pbl

  IBM employees may obtain LIST3820s of redbooks from this page.

- **REDBOOKS category on INEWS**
- **Online** — send orders to: USIB6FPL at IBMMAIL or DKIBMBSH at IBMMAIL
- **Internet Listserver**

  With an Internet E-mail address, anyone can subscribe to an IBM Announcement Listserver. To initiate the service, send an E-mail note to announce@webster.ibmlink.ibm.com with the keyword subscribe in the body of the note (leave the subject line blank). A category form and detailed instructions will be sent to you.

D.2 How Customers Can Get ITSO Redbooks

Customers may request ITSO deliverables (redbooks, BookManager BOOKs, and CD-ROMs) and information about redbooks, workshops, and residencies in the following ways:

- **Online Orders** (Do not send credit card information over the Internet) — send orders to:

	IBMMAIL	**Internet**
In United States:	usib6fpl at ibmmail	usib6fpl@ibmmail.com
In Canada:	caibmbkz at ibmmail	lmannix@vnet.ibm.com
Outside North America:	dkibmbsh at ibmmail	bookshop@dk.ibm.com

- **Telephone orders**

United States (toll free)	1-800-879-2755
Canada (toll free)	1-800-IBM-4YOU
Outside North America	(long distance charges apply)
(+45) 4810-1320 - Danish	(+45) 4810-1020 - German
(+45) 4810-1420 - Dutch	(+45) 4810-1620 - Italian
(+45) 4810-1540 - English	(+45) 4810-1270 - Norwegian
(+45) 4810-1670 - Finnish	(+45) 4810-1120 - Spanish
(+45) 4810-1220 - French	(+45) 4810-1170 - Swedish

- **Mail Orders** — send orders to:

IBM Publications	IBM Publications	IBM Direct Services
Publications Customer Support	144-4th Avenue, S.W.	Sortemosevej 21
P.O. Box 29570	Calgary, Alberta T2P 3N5	DK-3450 Allerød
Raleigh, NC 27626-0570	Canada	Denmark
USA		

- **Fax** — send orders to:

United States (toll free)	1-800-445-9269
Canada	1-403-267-4455
Outside North America	(+45) 48 14 2207 (long distance charge)

- **1-800-IBM-4FAX (United States)** or **(+1) 415 855 43 29 (Outside USA)** — ask for:

  Index # 4421 Abstracts of new redbooks
  Index # 4422 IBM redbooks
  Index # 4420 Redbooks for last six months

- **Direct Services** - send note to softwareshop@vnet.ibm.com
- **On the World Wide Web**

Redbooks Home Page	http://www.redbooks.ibm.com
IBM Direct Publications Catalog	http://www.elink.ibmlink.ibm.com/pbl/pbl

- **Internet Listserver**

  With an Internet E-mail address, anyone can subscribe to an IBM Announcement Listserver. To initiate the service, send an E-mail note to announce@webster.ibmlink.ibm.com with the keyword subscribe in the body of the note (leave the subject line blank).

D.3 IBM Redbook Order Form

Please send me the following:

Title	Order Number	Quantity

First name Last name

Company

Address

City Postal code Country

Telephone number Telefax number VAT number

- Invoice to customer number
- Credit card number

Credit card expiration date Card issued to Signature

We accept American Express, Diners, Eurocard, Master Card, and Visa. Payment by credit card not available in all countries. Signature mandatory for credit card payment.

DO NOT SEND CREDIT CARD INFORMATION OVER THE INTERNET.

List of Abbreviations

APA	all points addressable
DRDA	Distributed Relational Database Architecture
IBM	International Business Machines Corporation
ITSO	International Technical Support Organization
LOB	large object
MIB	management information base
PROFS	Professional Office System
RDBMS	Relational Database Management System
RFC	request for comments
SNMP	Simple Network Managment Protocol
SQL	Structured Query Language
UOW	unit of work

Index

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

Y

Printed in U.S.A.