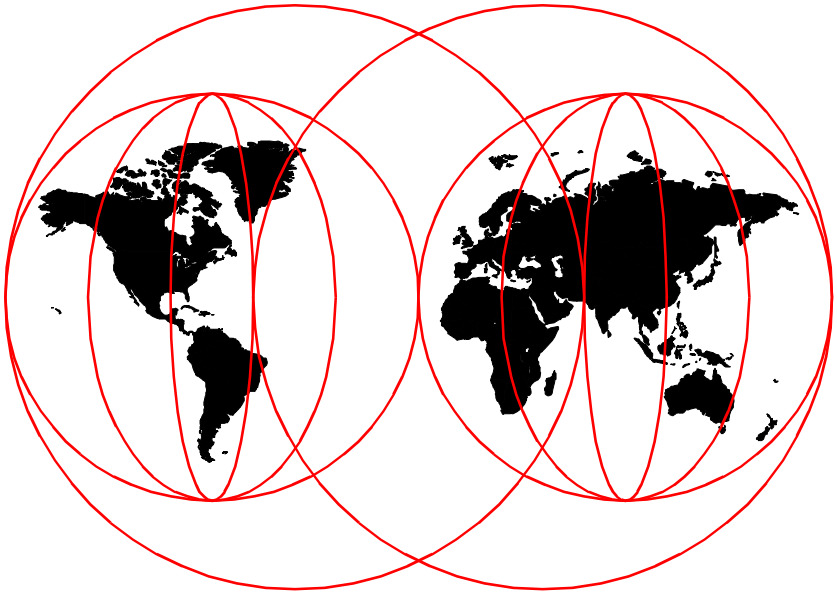
Database Performance on AIX in DB2 UDB and Oracle Environments

Nigel Griffiths, James Chandler, João Marcos Costa de Souza, Gerhard Müller, Diana Gfroerer



International Technical Support Organization

www.redbooks.ibm.com

SG24-5511-00

SG24-5511-00

International Technical Support Organization

Database Performance on AIX in DB2 UDB and Oracle Environments

December 1999

Take Note!

Before using this information and the product it supports, be sure to read the general information in Appendix E, “Special notices” on page 411.

First Edition (December 1999)

This edition applies to Version 6.1 of DB2 Universal Database - Enterprise Edition, referred to as DB2 UDB; Version 7 of Oracle Enterprise Edition and Release 8.1.5 of Oracle8i Enterprise Edition, referred to as Oracle; for use with AIX Version 4.3.3.

Comments may be addressed to:
IBM Corporation, International Technical Support Organization
Dept. JN9B Building 003 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

Preface

This redbook is designed to help system designers, system administrators, and database administrators design, size, implement, maintain, monitor, and tune a Relational Database Management System (RDBMS) for optimal performance on AIX. Relational Database Management Systems are a significant factor in the profit line of a company. They represent an important investment and their performance is often vital to the success of the company.

This redbook contains hints and tips from experts that work on RDBMS performance every day. It also provides introductions to general database layout concepts from a performance point of view, design and sizing guidelines, tuning recommendations, and performance and tuning information for DB2 UDB and Oracle databases.

How this book is organized

This redbook consists of three major parts, that are adopted to a database's life cycle.

The first part, *RDBMS concepts* contains information for a basic comprehension of RDBMSs, which is fundamental for understanding a database's performance behavior in order to design, size, and tune an RDBMS. This first part may be helpful if you want to learn about databases, as a reference to refresh your knowledge, or to help you understand why certain concepts have an impact on the database's performance. Different workloads are described in this part as well as the different architectures of DB2 UDB and Oracle databases and their parallel editions. System designers and system and database administrators will find this part especially useful.

The second part, *System design and sizing for optimal performance*, covers the time before the database is actually implemented and deals with preparing the system. This second part gives sizing techniques and provides a number of rules of thumb, from our experience, that can help you to size your database system. The design chapters deal with different circumstances that have to be taken into account when a database system is planed, such as growth or backup and restore needs. We dedicated one chapter to the disk subsystem design since this is the most sensitive and essential area for good database performance. This second part is vital for system designers and architects.

The third part, *System optimization*, covers the time from the database implementation on. First, we give hints and tips on how to optimize your

database implementation that go beyond the implementation documentation provided by database vendors. Then, we introduce some monitoring tools and methods that help you to keep track of your database's performance. The tuning chapters not only introduce different tuning methods and their benefits and drawbacks, they are, furthermore, filled with hints and tips and recommendations that help you tune the performance of your database system. There are AIX tuning hints and tips as well as separate chapters on DB2 UDB and Oracle tuning. The last chapter of this part should be used if all else fails. It provides hints and tips on how to open a problem record with IBM Software Service in a most effective way.

The appendixes are provided for your reference. They contain reference sheets for database administrators, AIX performance tools, vital SQL, and we describe the Model Database that we used for our performance tests during the development of this redbook.

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.

Diana Gfroerer is an International Technical Support Specialist for RS/6000 and AIX Performance at the International Technical Support Organization, Austin Center. She writes extensively and teaches IBM classes worldwide on all areas of AIX performance and tuning. Before joining the ITSO this year, Diana Gfroerer worked in AIX pre-sales Technical Support in Munich, Germany. She was leading the Region Central, EMEA, and World Wide Technical Skill Communities for AIX and PC Interoperability.

Nigel Griffiths is a Performance Guru in the RS/6000 Pre-Sales Technical Support Group in the UK. He has 20 years experience with UNIX, seven of which are with IBM. His areas of expertise include C programming including UNIX kernel internals, performance tuning, and sizing of SMP and parallel Oracle databases. He has been a performance and database technical support leader for six years and has written extensively on performance and sizing.

James Chandler is a Database Administrator with IBM Global Services in Lexington, Kentucky. He has four years of experience with storage management and database administration in distributed environments. He has worked at IBM for six years. His areas of expertise include implementing storage management solutions using ADSM and third party vendor tools, as well as database administration in SAP environments using Oracle and DB2.

Joao Marcos Costa de Souza is a DB2 UDB Support Professional in Sao Paulo, Brazil. He has seven years of experience in database administration and support in the field. He holds a degree in Computer Science and is an IBM Certified Solutions Expert - DB2 UDB V6.1 Database Administration and an Oracle Certified Professional Database Administrator. His areas of expertise include DB2 UDB EE and EEE implementation and support as well as performance and tuning.

Gerhard Mueller is a Software Engineer with AIX Software Support in IBM Global Services in Mainz, Germany. He has five years of field experience in DB2 on AIX. He is a Certified Solutions Expert for DB2 UDB Administration. His areas of expertise include installation, administration, and problem determination in DB2 UDB EE and EEE environments.

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

Elizabeth Barnes
International Technical Support Organization, Austin Center

Richard Cutler
International Technical Support Organization, Austin Center

John Owczarzak
International Technical Support Organization, Austin Center

Temi Rose
International Technical Support Organization, Austin Center

Tetsuya Shirai
International Technical Support Organization, Austin Center

George Accapadi
IBM Austin

Mathew Accapadi
IBM Austin

John Aschoff
IBM San Jose

Stephen Atkins
IBM UK

Richard Bridgman
IBM UK

Doug Doole
IBM Toronto

Jessica Escott
IBM Toronto

Ian R. Finlay
IBM Toronto

Angel González
IBM Germany

Andreas Hoetzel
IBM Austin

Karl Huppler
IBM Raleigh

Joey V. James
IBM Austin

Dale Martin
IBM San Francisco

Dennis Massanari
IBM Poughkeepsie

Sean McKeough
IBM Toronto

Walter Orb
IBM Foster City

Ram Pandiri
IBM Austin

Dr. Norbert Pistoor
IBM Germany

Steve Pittman
IBM San Francisco

Lilian Romero
IBM Austin

Berni Schiefer
IBM Toronto

Johnny Shieh
IBM Austin

Bill Topliss
IBM Austin

Aspi Wadia
IBM Austin

Eddine Walehiane
IBM Austin

David Whitworth
IBM Austin

Comments welcome

Your comments are important to us!

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

- Fax the evaluation form found in “IBM Redbooks evaluation” on page 443 to the fax number shown on the form.
- Use the online evaluation form found at http://www.redbooks.ibm.com/
- Send your comments in an Internet note to redbook@us.ibm.com

Chapter 1. Introduction to this redbook

Relational Database Management Systems (RDBMS) become more and more the core IT systems of companies and large enterprises. They are vital for companies' profit lines since RDBMS systems hold data, such as sales, stock, finances, and order income. This data has to be accessible by many people at the same time, often 24 hours a day, 7 days a week, especially since companies are extending their business onto a global market, and people need to have access to the systems from all time zones.
Not only production data for the daily business, but also historical data is held in the companies' database systems. This data is used for research and to provide information for major management decisions.

High performance of these systems is very often vital for the success of a company. Customer orders have to be processed quickly, and available stock in the warehouses has to be found and assigned to the according order. Especially for companies with time-critical data, such as airlines, good performance is mandatory.
Performance also becomes an increasing issue because the systems get bigger every year, the databases get more complex, and, last but not least, RDBMS systems mean a large investment in resources, both in money and people, and everybody wants value for their money.

Database performance is a very wide area with many different aspects. It is dependent on a large number of factors, such as the hardware, the application, the workload, the layout of the disk subsystem, and an uncounted number of system and database parameters.

Within this book, we want to share our knowledge and experience with you and help you to understand what database performance is all about and where to focus on when you plan, run, and tune a Relational Database Management System. We found that it was often hard to pin down clear facts since the alteration of one little parameter can change the whole picture, but we give you a number of *rules of thumb*, based on our experience, and we put a large amount of information down for you to make conclusions about the performance needs and impacts of your database system.

We also hope to cut through many myths and legends about performance tuning options that are no longer true or that are only part of the truth as well as give you an update on the latest features of AIX that you can use to make your database perform at its best.

You might find this book to be helpful in any stage of your database's life cycle: In the planning and sizing stage, during implementation, and when running a productive database system. We adapted the structure of our redbook to this life cycle and subdivided it into three major parts:

- RDBMS concepts - Covering the concepts of Relational Database Management Systems, the different workload characteristics, and an introduction into both DB2 UDB and Oracle databases, including a brief introduction into parallel database systems.
- System design and sizing for optimal performance - Covering the pre-life phase of an RDBMS the sizing to meet the requirements of the predicted workload, and the system design and layout for optimal performance.
- System optimization - Focusing on the implementation of an RDBMS and the monitoring and tuning tasks once the database is installed.

This book is written from an AIX and RS/6000 point of view and focuses on how an RDBMS can use the advanced features of these products.

Even though we are covering DB2 UDB and Oracle databases in more detail, a large part of the book also applies to any other Relational Database Management System. We chose DB2 UDB and Oracle because they represent 80 percent of all databases installed on RS/6000s, and 65 percent of the RS/6000 Enterprise and SP Systems run a version of these popular databases.

Database design or application programming are large subjects that are common to all platforms. There is a wide range of literature available on these subjects; therefore, we do not cover these subjects in this redbook, nor do we go into great detail on Structured Query Language (SQL). Please refer to Appendix F, "Related publications" on page 415 for some useful books. Appendix 3 has a simple quick reference sheet on SQL, useful if you are sitting in front of the machine trying to remember a particular SQL statement in order to get certain information from the database.

Parallel databases are briefly mentioned so that you know when to consider them. However, covering parallel database design and performance exceeds the scope of this redbook.

Designing, sizing, and tuning an RDBMS is rather an art than a science, and it requires a lot of technical skills and personal experience. This book, therefore, is a valuable source of information on your way of becoming a professional RDBMS performance expert.

Part 1. RDBMS concepts

Chapter 2. Introduction into relational database system concepts

This chapter covers the basic theory behind modern Relational Database Management Systems (RDBMS) and how they are implemented on AIX at an overview level. It also details what the RDBMS is meant to achieve and how it keeps the data safe. If you are new to databases, or have forgotten the basics and want a reminder, then this is a good place to start.

2.1 What is an RDBMS?

Data is information. A database is somewhere that you store data. Databases can store three different types of data:

1. Common data - Which can include numbers, dates, and strings of characters, such as names and addresses.
2. Complex and large objects - There are many esoteric data types that can be stored and managed by a database, such as sound, geographical data, such as maps, pictures, graphics, and even videos.
3. New user defined data - Most modern database systems also allow the user to store new data types that they define and want to manipulate.

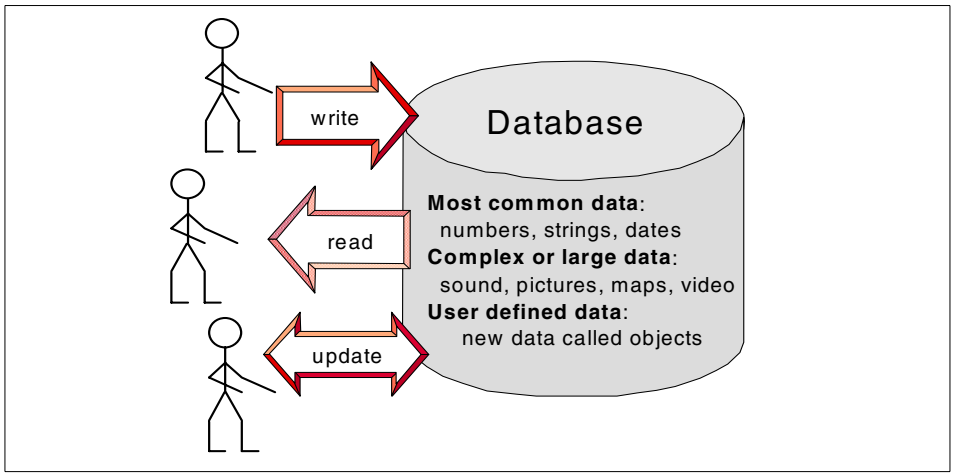


Figure 1. Databases write, read, and update many types of data

The database must allow the user to:

- Write information into the database
- Retrieve the information later

- Update the data

Databases must be able perform all these operations in a very reliable fashion (or we would not trust the database with important data) and at high speed. They also should be able to provide these functions to many people at the same time. As a side effect, a database can be used as a common place for information and provide many people with one *common* view. By this, we mean that if the database contains details of, for example, a parts inventory, everyone can see the *same* number of items that are available (there is only one true answer).

Although various types of data can be stored in a modern database, the vast bulk of production databases are used to store simple records of numbers, strings of characters, and dates. In this redbook, we concentrate on these data types.

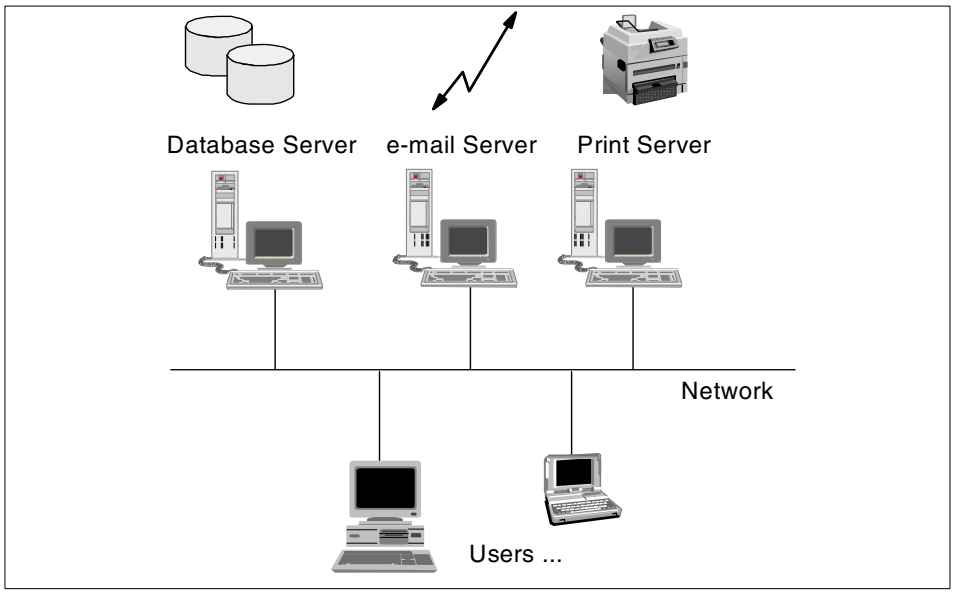


Figure 2. Classic UNIX approach of using multiple servers

In UNIX system environments, there is a tradition of each UNIX machine having one purpose. An example might be one machine as an NFS server, another as a printer server, and yet another as an e-mail server (see Figure 2). This is for a number of reasons:

- In the early days, UNIX machines were not as powerful as other systems (for example, the IBM mainframe). So, dedicating a server to the workload maximized the computer power for the workload.

- UNIX is strong on networking, which makes client server systems easier to implement; therefore, splitting workloads this way is natural for UNIX.
- Having the workload on different machines avoids interference between the applications. For example, if two compute intensive (or I/O intensive) applications on one machine compete for resources, the performance of both applications suffers.
- Limitation on the number of disks a single UNIX machine can support.

Traditionally, RDBMSs run on dedicated UNIX servers. This redbook assumes that most databases are running on dedicated machines. This means that the machine can be tuned for maximum database performance with no concern for other workload types. In the last couple of years, UNIX machines have become very powerful (by adopting many mainframe design characteristics). This results in large SMP UNIX machines on which a variety of workloads are run. An SMP machine is managed as one machine. The application workloads are, however, separated by logical or physical partitions to reduce competition between workloads for CPU, memory, or I/O resources. This means these machines perform like many smaller database servers joined together.

Note

In this redbook, we assume the RDBMS is running on a dedicated machine. Therefore, the system tuning for performance does not have to take other applications or workloads into account.

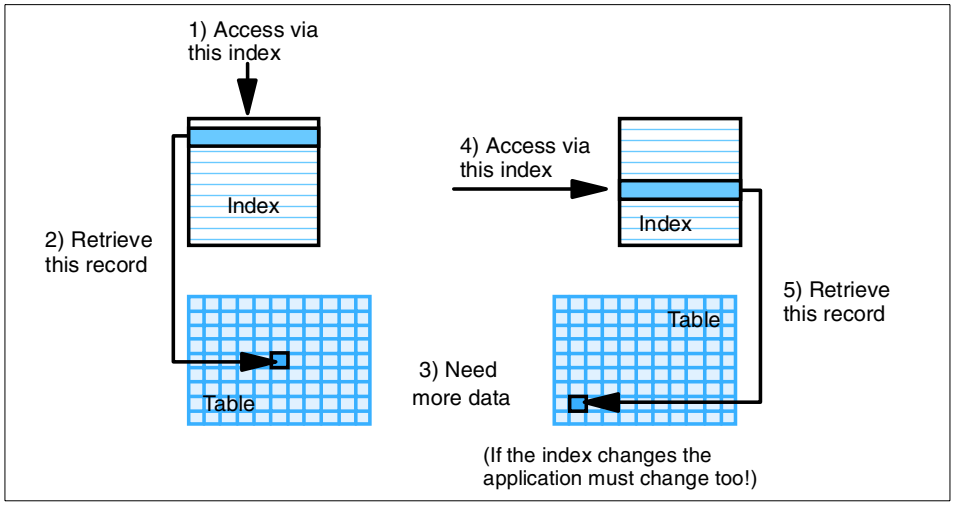


Figure 3. Non-relational Database Method

This redbook is about relational database management systems (RDBMS). What does the word *relational* mean? In a simple database, there are many records, and application can add more records to the database. This operation is called *write*, *put*, *save*, or *add* a record. They all mean the same thing. These records can later be retrieved. This operation is called *retrieve*, *read*, *fetch*, or *get* a record. Figure 3 shows how data is read from a non-relational database. To decide which record to retrieve, the application has to inform the database on how it wishes to *see* the records (access method, for example, the customer record in last name order) and the record identity (a number or name). If this record contains reference to other data (for example, the first record is for an employee, and it contains the department number), then the application has to include the code used to access these other records. It has to set up the access method and then read the record for the department to find out the department name, address, manager, etc. This is sometimes called a *one row at a time* application, which is inefficient.

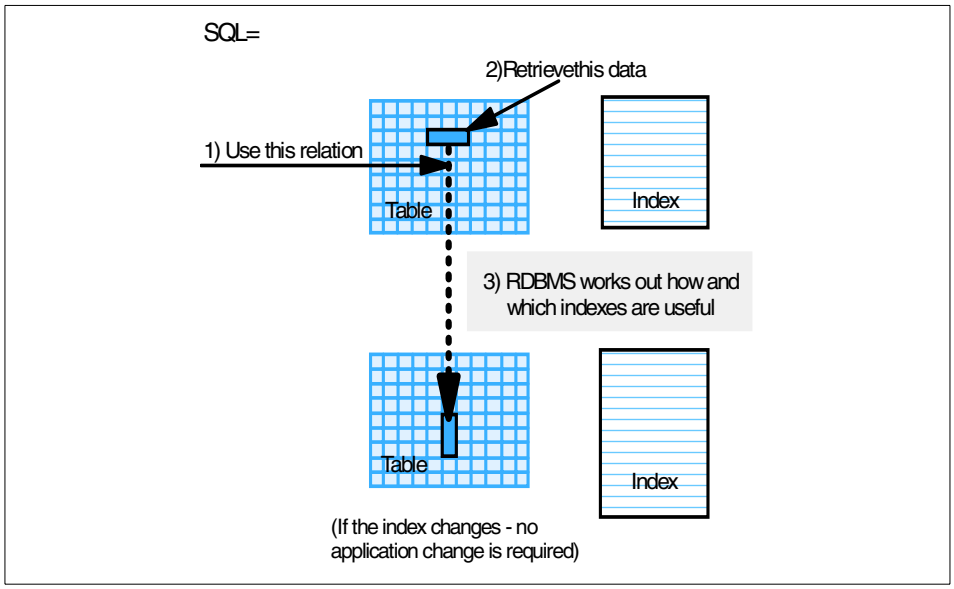


Figure 4. Relational database method

The alternative is the relational method used by an RDBMS. The database understands relations between data and uses this information to extract the data needed in a single operation. Figure 4 shows how RDBMS data is accessed. It allows the application to read the two records from two tables at the same time and only get back the relevant information it requested. The RDBMS has used the relationship to:

- Save the application making two requests.
- Reduce the information retrieved to what the application really needs (in other words, not the entire record, just the parts needed).
- Save the database from having to process two requests.
- Save the application from having to decide how to access the data.

That last part is probably the most important. In an RDBMS, the application does not need to understand how the RDBMS will access the data, just what data and relationship to use. This allows the RDBMS to intelligently make its own mind up on the best method. It also allows the database administrator (DBA) to change the database without effecting all the applications or requiring them to be recoded. Thus, one change from the DBA, for example, adding a new index, can increase the performance of all applications at a single stroke. The language used to describe what data the applications require and the relationship is called Structured Query Language (SQL). See

Appendix C.1, “SQL reference sheet” on page 381 for a summary and some details.

2.2 What does an RDBMS provide?

Maintaining a set of records is simple. We all could do this with a simple card index on our desk but imagine 200 people turn up at your desk and:

- 40 of them start adding new cards as fast as they can.
- 40 of them spot mistakes in your cards and start making adjustments to correct the cards.
- 10 start removing cards and throwing them in the waste paper basket.
- 50 of them start cross checking the cards to make sure things, such as addresses and telephone numbers, are correct.
- 40 of them are trying to update the same cards at the same time and start having heated arguments about who should have the card and if the last change was correct or not.
- 20 people have half-updated a card and went for lunch, and then someone else took the card and made other changes.

This then goes on day after day. Chaos would be the result as soon as:

- The number of cards occupy the entire room.
- The heated arguments can result in damaged cards.
- The details on the cards can not be trusted.
- Finding the right card would be increasingly hard and would take longer as the number of cards increases.
- If there was an accident (such as a dropped card, or worse, a fire or flood), we might never sort out the mess.

This is what an RDBMS copes with 24 hours a day. What we need is transactions that are ACID. What this means is:

ACID

- Atomic - Changes are fully completed or not done at all.
- Consistent - The data is never seen in a half completed transaction state.
- Independent - Ongoing changes are not allowed to interfere with each other.
- Durable - Once changed, the new data is guaranteed to be available.

These attributes are not simple. This is why people use one of the well known RDBMSs and do not write their own. How then are these attributes actually implemented in practice within the RDBMS?

- Atomic - This means that every change either completely works, and all records are updated, or the update completely fails, and the database is not changed at all. This is particularly important when you are considering things such a banking system. You should either transfer the money from one account to the other or not at all. A half completed change is not a satisfactory condition. The RDBMS system achieves atomicity by using a database log. If changes or updates are atomic, they are called *transactions*. When transactions make changes to the database, they are noted in the database log. When the outcome of a transaction becomes known (that is, it has finished or been abandoned), the RDBMS will update database records. Either all the records are updated, or all the records will be unchanged. To do this, the RDBMS keeps copies of the original records and the updated records in the log. The RDBMS uses the term *commit* to refer to a finished transaction and *abort* to refer to an abandoned transaction.
- Consistent - This is achieved by maintaining multiple, concurrent views of the same data. The main point of this attribute is that no program (or user) is able to see the database in a state between transactions. It appears to the user that a transaction takes an infinitely short period of time (which means the database suddenly changes at commit time). Why is this important? Without this attribute, it is possible to get misleading results. Think back to the banking example where, this time, a customer has $1000 in their deposit account and zero in their current account and is transferring all the money between these accounts as a transaction. At all times, the customer has $1000 in total. If the RDBMS was not providing a consistent view of the database, we could find the customer with (depending on the order of updating the records):

  1. $1000 (the right answer)
  2. $0 (that is, when the money is withdrawn from one account but not yet added to the new account)
  3. $2000 (that is, the money added into the new account before it is withdrawn from the first account)

  If one of the last two items on this list takes place for the program that prints monthly bank statements, then the customer could be very worried ($0) or delighted ($2000). Clearly, the database needs to provide consistency, and this is done by temporarily keeping multiple versions of records during transactions. Each transaction will see a consistent state of the data. If necessary, these copies are place in the database log.

- Independence - This is achieved by using clever locking mechanisms. A simplistic way of updating the database would be to allow only one program to do all the updates, but this would have severe performance limitations. So, an RDBMS must allow many programs to read and write records at the same time. Before updating records, a program locks the data for reading or writing so that no two programs actually update the same record at the same time. In a large database, there are millions or billions of records; so, the chances of two users wanting the same record is very low and, therefore, this works well. But, there are some records in the database that many users need. A classic example is the sales ordering processing database where the next invoice number is held in a single record. Every invoice needs to take the current number and increment it by one. These issues are well understood, and there are various methods to reduce the problem. These include RDBMS support for supplying simple numbers, the application taking a number and immediately committing to release the locks, pre-allocation of numbers (for example, one program uses the range of 1 million to 2 million and the next using 2 million to 3 million), and letting a background process do the actual updating.

  One problem with locking is that two programs can lock each other; therefore, both wait indefinitely for the other to release its locks. This is called a *dead-lock* or *deadly-embrace*. All RDBMS systems have mechanisms to spot this problem and to resolve it. Typically, the RDBMS will fail one of the transactions, and the application can (if coded properly) retry the update slightly later and, hopefully, after the other transaction has completed, it will not cause the same problem again.

- Durable - The data in the database is often vital for the business. If the transactions are lost, then major problems are expected. For example, in sales order processing, if the transaction information is forgotten, not only does the company not make a profit, but the customer gets annoyed when the goods do not arrive and may well take their business elsewhere. To make the transaction durable, we have to make sure that, whatever happens, the transaction's details are remembered. To provide atomicity, we used a database log. Durability uses the log too. When the transaction finishes, the outcome is also placed in the log (committed or aborted). If the database fails in some way (for example, a power cut or a disk crash), then when the RDBMS is restarted, it checks the database log. It will either find that the transaction finished, in which case the updated records are put into the database, or the transaction failed, in which case the RDBMS makes sure the original records are in the database.

Without these ACID attributes, your data is not in safe hands. Fortunately, these are the properties that an RDBMS provides. It is only through thinking about possible bad experiences of incorrect, damaged, or missing data that one can work out how important these attributes are for a database.

ACID attributes

ACID stands for:

- Atomic - changes are fully completed or not done at all
- Consistent - the data is never seen in a half completed transaction state
- Independent - changes are not allowed to interfere with each other
- Durable - once changed, the new data is guaranteed to be available

2.3 The database performance trick

The trick for an RDBMS is to provide these ACID attributes while maintaining high performance and scaling to large volumes of data. The main trick that RDBMSs use for performance is not at all obvious. To provide three of the ACID properties, the RDBMS uses the database log (the other property uses locks that are not logged because they are transitory). Normally, when a transaction finishes, this is noted in the log so that if a failure occurs the database knows the result and, if necessary, can rework the transaction. Once the log is written to disk, the transaction is saved so that the RDBMS does not actually have to immediately go and update the database records themselves. Instead, the database only writes to the database log and then tells the application (and user) that the transaction has finished. It does the updating of the data disks a little later and at its leisure. This method is used because the log and data disks have different characteristics. The records in the database are probably scattered all over many disks. This means updating them would take time, as the user would have to compete with all the other database I/O to the disk, and the disk heads are making large movements, which slows down access times (see Figure 5).

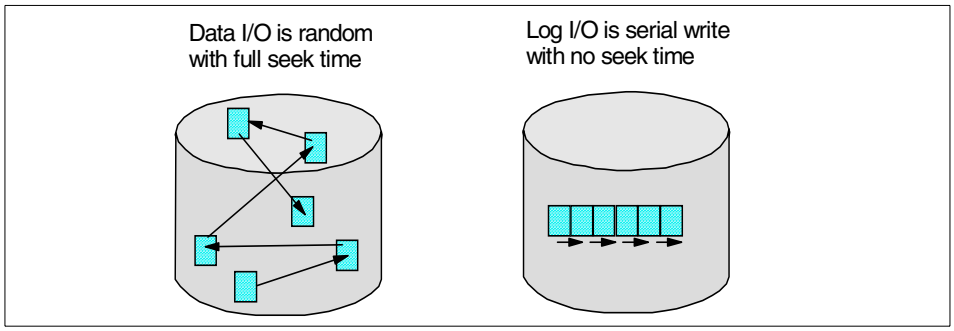


Figure 5. Data I/O is slow and random, but log I/O is fast and serial

The log, however, is written out as a sequential file, and if the log disk is dedicated to the log (which is normally the case), writing out to the log will be extremely fast. For the log disk, there is little or no disk head movement, and, thus, it has faster access times.

This RDBMS data logging has three side effects:

- Good performance - Transactions are finished very fast (before the records are even updated) and allow the user to continue with the next transaction.
- Logging performance is a focus point - The log is critical to the performance. This is why RDBMS logs are often on dedicated disks and on the fastest disks possible.
- The log disk is a failure point - As we have not updated the database records immediately, the log is the only place the recent updates are stored. If the log disk fails, we destroy the database because we no longer know which records where updated and which were not. We have lost three of the ACID properties. Most RDBMSs will refuse to allow access to the database until this log is recovered in some way. The solution to the single point of failure is to protect yourself from the disk crash by mirroring the log disk or by using RAID 5.

Note

The RDBMS log is the key to RDBMS performance and recovery.

2.4 What are the components of an RDBMS?

This section explains what to expect on a system that is running an RDBMS and what components provide what features.

The first thing you need for an RDBMS is somewhere to store the data, and that is on disks (occasionally also called DASD). Databases are getting larger every year, and most people now refer to the size of a database in GBs. The definition of *very large database* (VLDB) changes each year. Table 1 is a guideline for database sizes. But, note that this is the *data* size not the *disk* size. For disk sizes, use the 1:3 *rule of thumb* (for more information see Chapter 8, “Designing a disk subsystem” on page 149).

Table 1. Typical sizes of database

Database Description	Raw data Size
Minuscule or sample	Less than 1 MB
Experimental or test	100 MB to 2 GB
Tiny	Less than 1 GB
Very small	1 GB to 5 GB
Small	5 GB to 10 GB
Moderate	10 GB to 50 GB
Medium	50 GB to 100 GB
Large	100 GB to 200 GB
Very large (called VLDB)	200 GB to 300 GB
Extremely large	300 GB to 500 GB
Massive	Greater than 500 GB

Note, that for a good performance, there is a minimum number of disks required for an RDBMS. For the following example shown in Table 2, we assume that the disks are a few GBs in size, and the database size is very small. The system needs disks for different purposes as shown in Table 2.

Table 2. Minimum disk requirements

Disk use	Number of disks and comments
Operating System	1 disk.
Paging space	0 disks = Use above disk. 1 - 3 disks for larger memory sizes(2 GB or more).

Disk use	Number of disks and comments
RDBMS code	1 disk - To allow for upgrade of the RDBMS.
RDBMS data RDBMS indexes RDBMS tmp/sort	3 disks - It is recommended to use one disk for each of the three parts as a minimum (for databases that are less than the size of one disk, this could be the same disk, but this is likely to become a performance bottleneck).
RDBMS log	1 disk - Dedicated for performance.
Totals	6 disks is a minimum.

The conclusion is that, even for the smallest database, about six disks is the minimum for a well performing RDBMS.

In addition, extra disks would be required for mirroring or RAID 5 to allow full disk failure protection. Most databases allow the use of either file system files for the database or raw devices (that is, direct access to the disks by the RDBMS). The file system database is visible to all UNIX users, for example using the `df` or `ls` UNIX commands, but the files are not readable or writable by anyone other than the RDBMS processes.

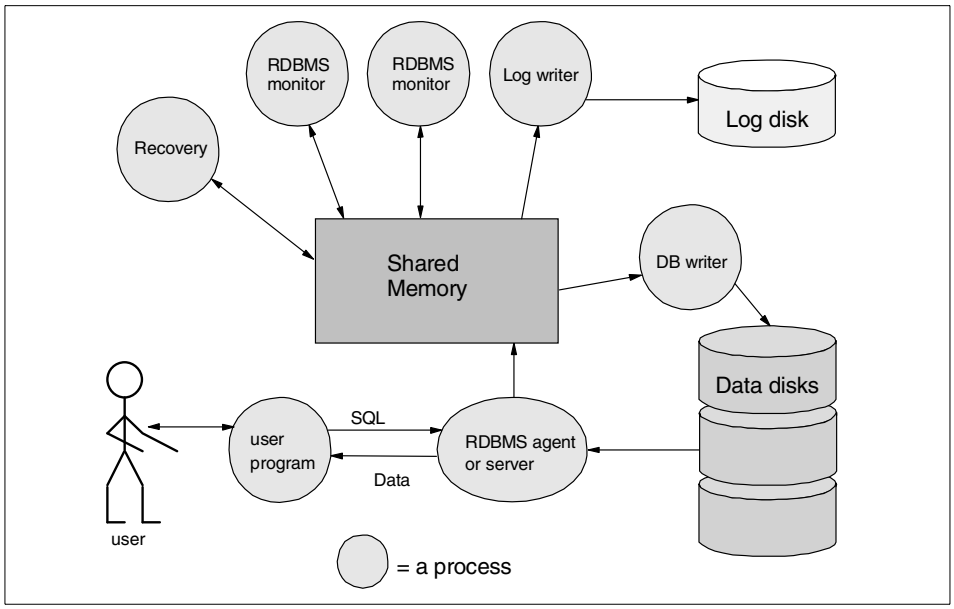


Figure 6. Structure of an RDBMS

All RDBMSs have to read and write data from the disks and into memory. But in computer terms, the disks are very slow when compared with memory. The access time to a disk is approximately 10 milliseconds, and the access time to memory is approximately 10 nanoseconds, that is, a ratio of one to a million (1,000,000). Therefore, to save time, the RDBMS keeps all the recently used disk blocks in memory. This is referred to as the RDBMS buffer cache or RDBMS disk block cache. This memory gives an RDBMS a major boost in performance. Ideally, the RDBMS would like to copy the entire database into memory. Unfortunately, memory is a lot more expensive than disks. The ratio is one to 40 for memory and disk space of the same size. This means, in practice, that only a small, but frequently used, part of the database is kept in memory. The RDBMS allocates memory using a *least recently used* (LRU) algorithm so that it dynamically works out the most important data. This memory on UNIX systems allocated by the RDBMS is called IPC Shared Memory and allows all the RDBMS processes to have concurrent access. A rule of thumb is that five percent of the database data is kept in memory. Also held in memory are a lot of internal data structures of the RDBMS itself. So, the shared memory includes:

- The RDBMS buffer/block cache
- The RDBMS lock data
- The log entries to be written to the log disk
- The SQL statement and query plan cache so that the RDBMS does not have to work out the best strategy for answering the same statement twice
- Some special tables used internally by the RDBMS

In practice, this memory is implemented as UNIX shared memory and is visible to the UNIX user via the `ipcs` command.

Note

The RDBMS Cache or pool is the key to performance by reducing disk I/O and allowing many users concurrent access to the same data.

The RDBMS code, data, memory, and processes have to be owned by a UNIX user. So, while installing the RDBMS, one of the first tasks is to create this user to own the RDBMS files, memory, and processes. Other normal RDBMS users can access the database but only if they have their permission set up to the RDBMS and only via RDBMS programs.

The users (or their programs, applications, tools, or code) are never allowed direct access to the RDBMS memory or files. This is because users are not

trusted. If a single user damages the database in memory database structures due to a poor program (or deliberately), then database data could be corrupted, and the database could crash. User programs (from any source) can only interact with the RDBMS via a special library, which, itself, interfaces via a RDBMS supplied process that is connected to the RDBMS memory and files. This seems like over-kill, but user level RDBMS programers are famous for making mistakes (particularly with pointers in the C code), which can compromise the RDBMS.

Each RDBMS has a set of processes that provide the core function of the RDBMS. These processes have to do a number of things:

- Recovery - recover the database when it fails. This is actually done on the next database start.
- Sanity checking - Monitor the RDBMS to check for major problems.
- Clean up - Monitor transactions and user processes and, if they fail, put the record back to its original state and remove the locks.
- Log writing - Write the database log to disk whenever a transaction finishes.
- Database update - Write the updated data and indexes from memory back onto the database disks.

Each RDBMS uses different names for the various parts. In DB2 UDB, all the processes are named db2<something>, and in Oracle, the processes are named ora<something>.

Note, that the processes described above are doing RDBMS house keeping.

So, which processes actually do the work for the user in terms of:

- Executing the SQL statements?
- Reading data from the disks into memory?
- Returning data back to the user program?

When a user program starts up, it has to make a connection to the RDBMS. We said before that user programs are not allowed to directly use the RDBMS resources; so, during the connect, the RDBMS starts or allocates a special RDBMS process to do the work requested by the user program. These special programs are called differently by the various RDBMS vendors. For DB2 UDB, they are called DB2 agents, and for Oracle, they are known as Oracle servers, background servers, or Oracle slaves. Until the user program stops, crashes, or disconnects from the database, these RDBMS processes are dedicated to this user program.

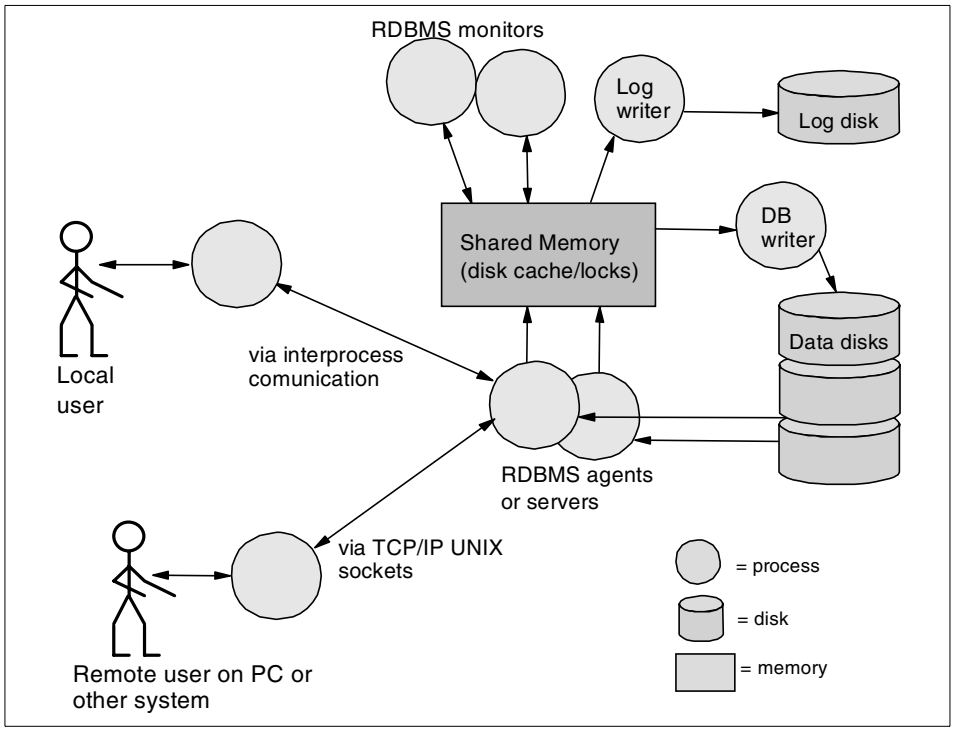


Figure 7. Local or remote user access

Users can actually be using local dumb terminals, accessing the RDBMS from another system, or using a PC. This connection to an RDBMS can be local or remote. The same thing happens in both cases, but the communication method used it different. Typically, local connections use UNIX Inter Process Communication (IPC) services, and remote connections use UNIX sockets over TCP/IP. Whichever method is used, the user ends up communicating to the RDBMS via the DB2 agent or Oracle server.

UNIX IPC Services

UNIX IPC services are shared memory, shared message queues, and shared semaphores. These are used to communicate between processes on the same machine at high speed.

Users are quite often not good program developers. So, how do non-technical users get access to the data that is stored in the database? First, they can use applications written for them. These may be written by their company, or they might be bought in from third parties. Either way, the user either starts

the application or starts a graphical interface that communicates with the application.

For database administration (and for experienced users that can write SQL), there are actually applications written by the RDBMS vendor. Before the RDBMS is started, and to control the RDBMS once started, there is a Database Administrator (DBA) tool. This tool (or group of tools) allows the DBA to create the initial database, start the database in exclusive mode (only the DBA has access), make large changes to the database and the way it operates, and, of course, shut the database down.

There is one final important tool available to the DBA and to normal database users (if given the permissions to run it). This tool has the generic name of the database *interactive monitor*. For DB2 UDB, it is the `db2` command, and for Oracle, it is the `sqlplus` command. These tools allow the user to type in SQL statements directly, send them to the RDBMS, and output the results in a reasonably sensible format. This allows the user to:

- Type in an SQL statement and run it (without writing a program every time)
- Experiment with SQL statements for education
- Do ad-hoc SQL to answer specific questions from the data
- Try alternative SQL statements to determine the one with the best performance

These tools also have non-SQL features that control the output format, provide response time information on the SQL, and output details about the database, tables, and indexes. In DB2, the UNIX shell can be used to provide programming features to the basic SQL statements. The Oracle `sqlplus` tool also provides a complete programming language PL/SQL (like BASIC with SQL added).

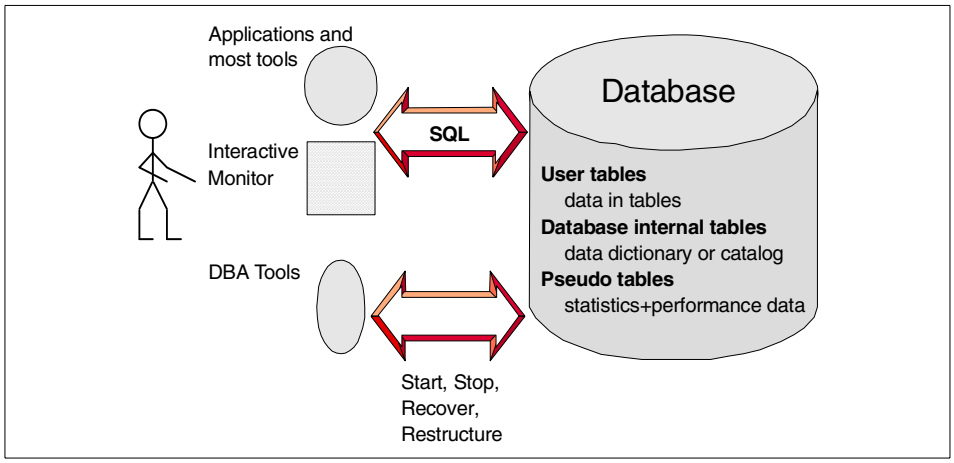


Figure 8. User tools and table types

Figure 8 shows the various methods of interacting with the RDBMS.

The database has the tables and indexes for the users. But how does RDBMS keep track of all the table names, column names and types, the indexes, user details, and structures? The answer is simple, the database has yet more tables that describe the user tables, indexes, and users themselves (names, passwords, and privileges). It also has many internal tables used to control and monitor the RDBMS while it is running. In effect, it has a database about the database. This is called a catalog or dictionary in all RDBMSs. In DB2 UDB, it is called the *catalog*, and in Oracle, it is called the *system data dictionary*. Some of these internal tables do not actually exist as tables on the disk but are internal, temporary, and dynamic variables within the RDBMS about the currently running system. The RDBMS gives the DBA and database user an SQL interface to this data so that there is one interface to all the data. This is a mandatory feature of a relational database, which states there should not be any other data access method to internal data. Therefore, each vendor's interactive monitor gives both the user and DBA one consistent interface to this internal RDBMS data, the RDBMS internal table structures, and the user's own data tables. This makes this interface a very important tool.

There are yet more tools with RDBMS, but each has a specific purpose. For example:

- A data loading tool to rapidly load vast quantities of data and as fast as possible. These can either use SQL to load the data or bypass the RDBMS and load directly into the databases files.

- Backup and recovery tools to save the contents of the database and reload it. There is often various methods and options that balance safety against speed and may interface with tape management software, such as IBM ADstar Storage Manager (ADSM).

Note

The database administrator has many tools to use and learn. Effective usage of these tools is vital for a safe database and good performance.

These tools are very specific to each RDBMS. Some are covered in more detail in Chapter 10, “Monitoring an RDBMS system for performance” on page 203.

2.5 Defining the RDBMS terms and ideas

In the previous section, we have deliberately been a little vague in the use of terms describing the ideas behind the RDBMS. In this section, we will define the terms more accurately.

2.5.1 RDBMS terms

RDBMS transaction

This is a unit of work that is either committed or aborted. This means the changes required are either completely done or no changes are made at all. In an RDBMS, a transaction is automatically started whenever the user (actually their application) performs any `SELECT`, `INSERT`, `UPDATE` or `DELETE` SQL statement. At the end of the transaction, the user has to use the `COMMIT` or `ABORT` SQL statement. In the `COMMIT` case, the RDBMS logs the transaction, the changes are made, and the lock is released. If it is `ABORT`, then the RDBMS undoes any updates made so far, logs the failure in the database, and releases the locks. Transactions are assumed to fail unless the `COMMIT` is found in the log. Most transactions `COMMIT`; so, the RDBMS attempts to make this happen as fast as possible.

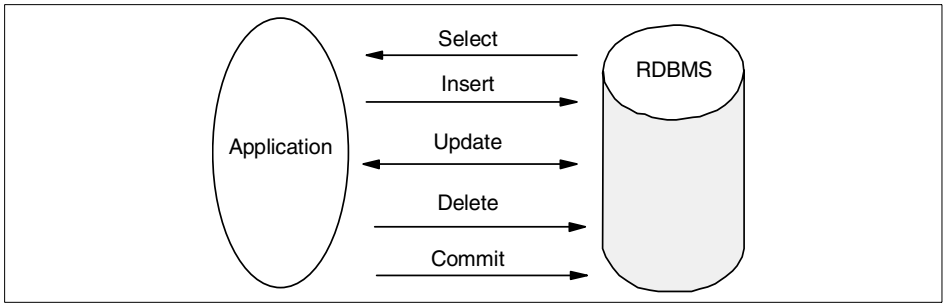


Figure 9. RDBMS transactions

Business transaction

Business transactions are often confused with RDBMS transactions. These are the transactions as the user views the system. A business transaction might be creating an invoice. Depending on the application, this may actually be many RDBMS transactions. The initial filling of the user's screen might be an RDBMS transaction to extract basic data like the customer details. Further RDBMS transactions might be used to look up data, such as part numbers or supplier details. Finally, when the invoice details are finished, and the user hits the *commit* key, the RDBMS will do a transaction to save the new details in the database. Applications use many small RDBMS transactions during one business transaction because this reduces the length of time locks on the records are held.

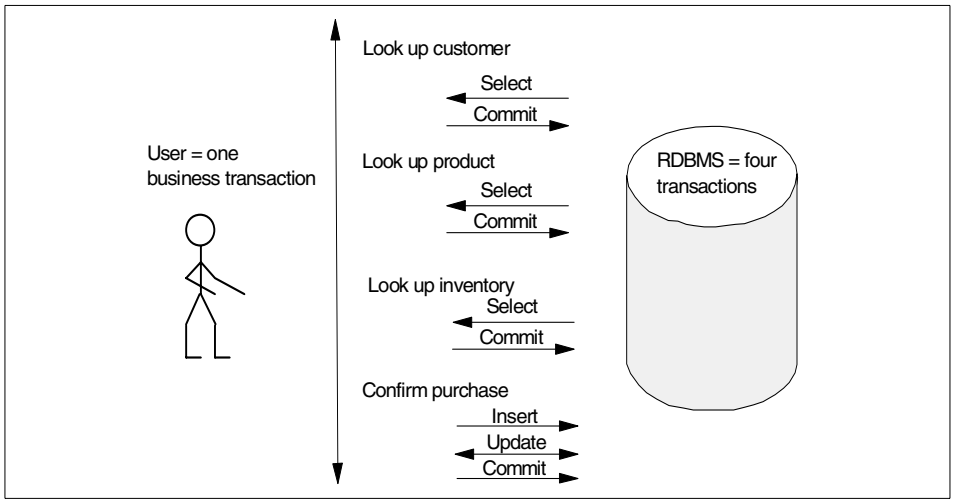


Figure 10. Business transactions

Commit One possible result of an RDBMS transaction, which saves the data or updates to the database. See *RDBMS transaction* and the other result, which is *abort*.

Abort One possible result of an RDBMS transaction, which removes all changes of this transaction. In the database, nothing will have changed after the abort. See *RDBMS transaction* and the other result, which is *commit*.

Instance This term has two meanings depending on the RDBMS. For DB2 UDB, it means the physical database data and files. For Oracle, it means the set of RDBMS processes that are connected to the database.

RDBMS This term includes the database data, files in which the data is stored, the running database processes, and the database shared resources, such as shared memory, the code, and tools.

Shared Memory This is memory on the machine that is dedicated, in this case, to the RDBMS. All of the RDBMS processes have access to this memory and mainly use it to store copies of disk blocks and control information. The RDBMS processes cooperate in using the memory and use locks to control access.

Buffer Pool or Buffer Cache This is the part of Shared Memory used to store copies of disk blocks to update them and to speed up processing. Buffer pool is mainly a DB2 UDB term, and Buffer cache is mainly an Oracle term, but the use of these terms is often intermixed.

SGA System Global Area. This is the Oracle term for all the information in Shared Memory.

Tables All of the information in the RDBMS is stored in tables. This concept closely matches the common spread sheet. The table has *rows* and *columns*. The columns are the various attributes of the data, for example, the name, the address the telephone number, and a date.The rows are the data items. These are similar to records (rows) and fields (columns) in non-relational database files (tables).

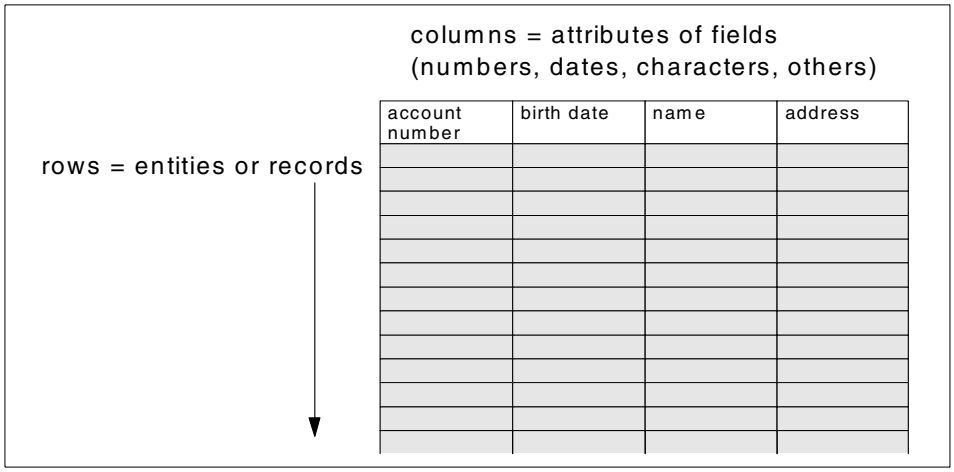


Figure 11. Tables, rows, and columns

Row

Rows contain the data of the database. In relational theory, they are called entities, and in a non-relation database, they are called records. See also *tables*.

Column

Columns are the various parts of a row. Each row of a table has the same number and type of columns. A column can be of only one type of data, such as numbers or characters. Columns in relational theory are called *attributes of the entity,* and in non-relational databases, called *fields of a record*. See also *tables.*

Relation or Relationships

This is a term used to describe how the RDBMS should connect two tables in the database. The relationship is made between two columns in the two tables. Typically, each table has a special column that is unique for each row (see *keys*). When one row refers to a row in the other table, it has this unique identifier in one of its columns. The relationship is described in the SQL statement by referring to these two columns. It should be matched up in the *where clause*.

View

A view presents one or more tables in a way that makes the view seem like a new table. The view does not contain any data, as the data is still in the original tables. A view can also be thought of as a pre-defined SQL statement that can hide the underlying table's details and relationships. A user or application cannot tell the difference between a table and a view.

Keys

Columns used to build relations between tables are said to contain keys. SQL relations are created via these key columns. Do not confuse these with index values, as SQL does not use indexes directly. There are two types of keys - primary and foreign.

Primary key

To be able to refer to a particular row of a table, there needs to be something unique about it. Database designers usually add a column to every table specifically for this purpose. These numbers or character strings are called primary keys. When a row wishes to refer to data in a row of another table, all that is needed is the primary key. This reference to the primary key is called a *foreign key*, (see foreign key). The primary key is a good candidate for an index because many SQL statements will often specify a particular row using this key.

Foreign key

This is a column of a table that refers to the primary key in another table so that the tables can be joined in an SQL statement (see *primary key*).

Multi-part key

Sometimes the primary key is made from more than one column. Provided the columns together uniquely define a row, this combination can be used as a primary key.

Security

The RDBMS has a complete security system with users clearly defined to have access rights to particular tables. There are SQL statements to define a user and grant and revoke privileges. This redbook does not cover security in any further detail, as it is not a performance issue.

Indexes

In the SQL, language indexes are not mentioned in the data access statements of `SELECT`, `INSERT`, `UPDATE`, and `DELETE`. A RDBMS will run without any indexes. Indexes are added to tables by the DBA to speed up finding particular rows in large tables. A table can have more than one index, and an index can be on one or more columns.

Temporary Storage or Tmp Area or Sort Area

When an RDBMS is performing indexing, sorting large tables, or preparing the results of SQL statements, it quite often needs to hold very large volumes of data. If this does not fit into memory, then the RDBMS temporarily stores the data on disk. This will slow down the processing but still provide the possibility to eventually finish. This area on disk is large. It can be as large as the raw data size of the database; so, it represents a large proportion of the disks.

Logs The database uses the log to save the updates to the database data. It also provides disk crash protection and good performance.

Locks To make sure the two processes or transactions do not attempt to update the same information at the same time the database uses locks.

Optimizer An SQL statement details what is required, but the RDBMS optimizer has to work out how to get and organize the data. There is often many different access methods including the order of reading tables, sorting method, and data merging strategies. The optimizer decides which is the most efficient. This plan takes time to workout; so, the RDBMS caches the query plan to save time if the same query is requested again.

Structured Query Language (SQL) terms

SQL is a standard syntax for accessing relational database systems based on a relational theory, which is documented in *The Relational Model for Database Management* by E. F. Codd. This book covers the 12 rules that database management systems need to follow in order to be described as truly relational.

The following are SQL statements or terms:

`SELECT` An SQL statement to read data from the database.

`JOIN` The relation used to connect two tables so that data can be extracted. The two tables are joined to form one large logical table.

`WHERE` ***clause*** Part of SQL used to specify a join or a restriction.

`INSERT` An SQL statement to add new data to the database.

`DELETE` An SQL statement to remove data from the database.

`UPDATE` An SQL statement to modify data already in the database.

SQL functions SQL provides standard functions that can be used in SQL statements. Most RDBMSs add other non-standard functions to make SQL more useful.

Aggregates	These are SQL standard defined functions that perform useful operations on groups of rows returned by the RDBMS. These include `sum()`, `avg()`, `count()`, `max()`, and `min()`.
`CREATE`	An SQL statement to make something in the database, such as a table or index. The RDBMS vendors make additions to the standard SQL. These allow creating of the database itself and to create tablespaces and offer fine tuning to database structures and resources. These additions are not standard between vendors.
`DROP`	An SQL statement to remove a table or index completely.
Tablespace	This is the place where tables and indexes are created. It is not part of the SQL standard, but DB2 UDB and Oracle both use this concept. The DBA creates tablespaces from files or raw devices in the UNIX system and then creates a table within it. This allows the DBA to place tables and indexes onto particular disks or sets of disks.
Sub-query	The answer to one SQL statement can be used within the *where clause* of another SQL statement. This is called a sub-query.
Relational operators	=, >, <, =>, <= used in an SQL statement.
Logical operators	`AND`, `OR`, or `NOT` used in an SQL statement.
Singleton select	A `SELECT` statement that returns one row.

2.6 Structured Query Language

The relational database is based on the use of the Structured Query Language (SQL). Early work was performed by IBM, but SQL is now a formal international standard to which most RDBMSs conform. SQL provides the core language to access data from an RDBMS. The prime statements in the language are the Data Manipulation Language (DML) statements:

- `SELECT`
- `INSERT`

- UPDATE
- DELETE

These statements give access to the data within the database. There are also Data Definition Language (DDL) statements, such as:

- CREATE TABLE, CREATE INDEX, CREATE VIEW
- DROP TABLE, DROP INDEX, DROP VIEW

GRANT and REVOKE (used to control user access to data)

Each database has added to the standard basic statements for the following reasons:

- To make the language easier to use (extra functions for common SQL tasks)
- To reduce programming effort (to save time and reduce the amount of coding required)
- To clarify certain ambiguous parts or options in the language (for example, outer joins)
- So that it can also be used to control the databases behavior itself (creating roll back areas and logs)
- For performance optimization (such as parallelization and optimizer control)
- To be very specific on data placement and space use (such as table defaults)

To make matters confusing, some extensions are simply new SQL statements, but others add new options to the standard statements. Typically, programmers can (if careful) avoid RDBMS specific extensions and, thereby, make their application portable between RDBMSs. The DBA, however, is forced to use the extensions because these are used for data placement on disks and for performance tuning. The DBA also uses very RDBMS specific options and internal database parameters for tuning the RDBMS.

Please refer to Appendix C.1, “SQL reference sheet” on page 381 for a quick reminder of SQL syntax and examples of SQL.

If you are looking for an introduction to SQL or an advanced manual, please refer to Appendix F.3, “Other resources” on page 415 for some excellent references to whole books on the subject.

2.7 How do we make the data safe?

Based on business needs and available budgets, every customer has to determine how safe they need to make their database systems. A helpful way of viewing the important decision is - What is the cost of the RDBMS not being available? Many businesses cannot survive without their computer system because it controls many operations of the company. Some industries start loosing money if the system is down for more than two minutes, for example, airlines and telephone ordering systems. Some businesses can run manually for a few hours or a day or two. Other systems are mainly back room activities and not visible to customers or directly involved with the company's finances. For example, some Business Intelligence systems do not stop sales when they are not available.

Another important factor is - What is the damage of losing data? For sales systems, this means lost business and angry customers. Some systems capture data that can never be replaced, and on other systems, wrong decisions may be made based on inaccurate data.

Modern computers are very reliable and get more reliable all the time. But, occasionally, they do have failures. The most common failures of RDBMS systems on AIX are:

1. Temporary power loss to the system.
2. Network goes down.
3. Disk crashes.
4. AIX system administrator makes a mistake or has problems with something that should work, such as upgrading software.
5. RDBMS administrator makes a mistake, for example, dropping a table.
6. Hardware failure in CPU, memory, adapter, or motherboard.
7. Total site disaster, such as fire, flood, bomb, hurricane, tornado, and so on.

What can we do about these problems, even if they are very unlikely? First, every RDBMS site must determine a policy regarding the down time that is acceptable. This can range from 60 seconds to five days. This, on the other hand, affects the time and money that is spent on making the RDBMS systems really safe. For each of these problems, there are options to remove or reduce the impact. First, assume we have no disk protection (how to protect disks is covered later).

1. Power loss - This is the most common problem with any computer and can be caused by accidental pressing of the off button or removing the wrong

power plug, fuses, or whole site power loss. Whatever the cause, once power is returned, the machine restarts automatically, and the RDBMS will also automatically recover the database up to the last committed transaction. The database recovery time depends on the volume of transactions on the system. This can be from a few minutes to a few hours in the worst case. Using an uninterruptable power supply can avoid this problem or at least give a period of time to stop the system cleanly.

2. Network - This means the system is unavailable, but once the network is fixed, the RDBMS is still OK. The only counter measure is alternative routes between users and the RDBMS server and good management practices. Networks are typically controlled by a different group of people than the database specialists. A network failure means the RDBMS is, strictly speaking, not available to users and can, therefore, be regarded as a lack of RDBMS service.
3. Disk crash - This affects the RDBMS directly. Assuming there is no disk protection, such as RAID 5, mirroring, or a standby system, then the damage depends on which disk is faulty. If the faulty disk is a:
   - Data disk - The disk can be replaced and a backup (see Part 2.8, “Backup and performance” on page 35) can be recovered onto the disk. When the RDBMS is restarted, it will replay the RDBMS log and, in effect, do all the transactions again since the backup. This will bring the replacement disk back up-to-date. This might, however, take a long time and depends on having an available disk, the speed of recovering the backup file, and the number of transactions the database has to recover.
   - Index disk - There are two choices for index disks. First, this can be treated like a data disk and recovered in the same way. The alternative is to replace the disk, inform the RDBMS that the index has been destroyed, and then re-create the index from the data. The time taken to re-create the indexes depends on the number of indexes and the size of the tables. Usually, which ever is the quickest method is used, but it can be very hard to determine which is fastest.
   - Temporary or Sort Area disk - This data does not need to be recovered. There is no data on these disks that the database needs. The RDBMS should restart with this disk missing or after the DBA has informed the RDBMS to ignore the disk. The DBA should try to find alternative disk space to make up for the missing temporary or sort area space.
   - RDBMS Log disk - This is a disaster. Without the RDBMS log, the database cannot be restarted. All recent transactions are lost forever. The only option is to reload the entire backup. If older parts of the load (since the backup) were copied to other disks, they can be used to

recover some of the transactions. This is why the RDBMS log should always be the first to have some sort of disk protection. If the log cannot be recovered, then the only alternative is to go back to the last backup.

- Operating system, RDBMS configuration files, RDBMS code, or application code disk - These must be recovered from the system backup.

4. AIX system administrator - To try to stop this from happening, system administrators should be well trained and maintain high skill levels. Nothing replaces experience in running large production systems. All operations, such as updating the AIX system, RDBMS code, or application code, should be tested on other systems and only implemented after full backups and out of normal working hours. Sites that make a lot of changes to their system suffer a great deal more than those with tightly controlled update schedules and methods. If this machine is part of a High Availability Clustered Multiple Processing (HACMP) or replication cluster, then the alternative backup machine should spot the failure, and the service is resumed a few minutes later.
5. RDBMS administrator - DBAs do make mistakes, and a full and tested RDBMS backup system is the only precaution that can help. HACMP would not help because both the machines share the now corrupted database. Unfortunately, a replica system would not help in this situation because the mistake will be replicated to the other machine, and both the machines now have corrupted databases. A backup is needed to recover from this problem. The database then needs to be recovered to the time just before the corruption.
6. Hardware failure - This failure is unlikely to corrupt the database; so, once the system is repaired, RDBMS will recover the database quickly. The time depends on the number of transactions in the RDBMS log. An HACMP or replica configuration would mean a takeover to the duplicate machine while the failed system is repaired.
7. Total site disaster - A number of things are vital to recover from this problem.
   - A full recent off-site backup
   - Clear documentation on the way the system was configured
   - Alternative hardware and network
   - The skills to do the job

Note that some transactions will be lost. Alternatives to avoid this problem are using geographic, high-availability solutions, such as HAGEO, remote disks, such as the SSA disk sub-systems can allow, or remote replication.

From the above, three important facts should have become clear:

1. Disk protection

   One important benefit of AIX is that it does monitor disk errors and reports them to the AIX system error log. Disks tend to start reporting intermittent errors before they actually fail completely. A good administrator should be monitoring the AIX error logs, and if this is being reported, actions should be taken immediately to prevent loosing data. With AIX, logical volume data can be migrated to alternative or spare disks with the system up. This means the disk can then be replaced at a convenient time.

   But, some disk do fail without warning or before data can be saved. Therefore, it is highly desirable for a reliable RDBMS that the disks are protected in some way. There are various alternatives:

   - Mirroring - Good for performance but adds extra cost.
   - RAID 5 - Lower cost but also lower performance if the fast write cache option is not used.

   These are discussed in more detail in Chapter 8, “Designing a disk subsystem” on page 149.

2. Standby machine

   With a database system, there are two alternatives for a standby service to recover in a short time from many of the above problems. Both of these require duplicate hardware:

   - HACMP can provide alternative CPU, memory, and network resources but use a shared disk and database (no duplicate disks are required). This IBM AIX product is a market leader in monitoring the status of a cluster of machines and automatically taking over workload, disk, network addresses, and services when a problem is detected. HACMP does require careful setting up and testing but will automatically recover from many of the problems above.
   - RDBMS replication is a service on a duplicate machine and a duplicate database. The primary RDBMS sends all local database updates to the replica RDBMS so that the two databases are the same. In practice, the replica will be slightly behind the primary database due to the time it takes to complete replication. Note there is a performance cost, as replication adds significantly to the

workload of the primary RDBMS. Every row that is changed is also copied to a replication table. From there it is read by the replication process and sent on the network to the replica. When the replica confirms it has done the update, the entry can be deleted. This can double the workload on the RDBMS server. RDBMS replication is available from the RDBMS vendor.

3. The database backup is vital for recovering disk crashes.

   Having a regular backup of the database is not optional but mandatory. There are many ways and options to perform a backup. We list and comment of many of these below and then recommend the best approach. Note that deciding the full details of a backup strategy and method, and then testing it, is a large amount of work and often left until too late.

Table 3. Making your RDBMS safe from common problems

Problem	Impact	Precautions
Power	low	UPS
Network	low	alternative routes
Disk crash	high	disk protection and backup
UNIX system administrator	high	backup and HACMP
DBA	high	backup
Hardware	high	HACMP or Replication
Site disaster	high	backup

2.8 Backup and performance

It might sound strange to consider backups and performance at the same time. Unfortunately, taking a backup either requires the database to be stopped, in which case, there is zero performance, or the backup is performed with the database running, and the backup will have a large performance impact. In this section, the various options are detailed.

Before the backup, the data is on the database disks. The backup process involves getting a copy of the data to some other media that is reliable, inexpensive, and moveable to an alternative site.

2.8.1 Backup media

There are many options in backing up media:

Tape This is the classic backup media. Tapes are inexpensive and can hold a large volume of data in a small package. The RS/6000 has the typical UNIX tape formats, such as 4 mm and 8 mm DAT tape drive and 9 track tapes. But it can also connect to the tape drives commonly used on AS/400 and mainframe machines that can offer higher performance in terms of throughput. Also, multiple tape drives can be attached to reduce backup time and also offer redundancy in case of a problem with a tape drive. The best tape drives currently available can match the speed of disks in their data transfer rates.

Disk One option is to back up the database disks to a different set of disks. This means an extra expense of more disks but can reduce the time the backup takes significantly. Once the backup is complete, the database can return to normal operations. The disks are then often backed up to tape or other media. If the database disks fail, this extra disk copy is very convenient because the data is immediately available for a recovery to start. Care must be taken to maximize the performance of these disks, or the disk to disk backup can take a long time. As this is a copy of the real database, many sites do not use disk protection on these backup disks. Some sites even move the disks to another site.

Mirror breaking If the database disks are using mirrors for disk protection, then one copy of the data can be made by splitting the two copies of the data. The original is still part of the database, but the copy can be used to perform the backup without effecting the database disk performance. There are two problems with this. First, if the database had a mirror for disk protection, then when the mirror is split off, there is no longer any disk protection. This means most sites use a three way mirror and split the third copy off, and the two remaining will still give disk protection. Secondly, after the mirror has been used as the source for the backup, the mirror has to be rejoined to the original disks. This is called *resilvering*. As all the disk blocks of one disk have to be read from the original and written to the resilvered mirror, the disks will be very busy for a long period of time. This will significantly affect database performance but is often forgotten. Mirror breaking is used to reduce the time the database is not available.

Network The database can be backed up over a network. Many sites have a collection of tape drives connected to a backup and tape management system. The database system sends the data across the network. This assumes the network is available, can be dedicated to the backup, and can provide the bandwidth requirements for the backup. The backup system will eventually place the data on the backup media. Often, they temporarily store the data on internal disks as a staging area before writing to tape at high speed.

Optical Some sites have legal requirements to archive data for many years and achieve this by using optical storage for their backups. Optical storage has a reputation of not being very fast.

2.8.2 Full or partial backup

Usually, you want to back up the whole database and the whole system in a *full backup*. This means that one set of tapes, for example, contains the whole computer system: Operating system, applications, RDBMS code, and data. But backing up the entire system means a large volume of data and the maximum backup time.

One way to reduce backup time is to reduce the data volume that is backed up every time. This is called a *partial backup*. For example, during the week, one fifth of the database is backed up every day with a full backup only once a week. This means, though, that the recovery time might be a lot longer (because the weekly and partial backups will need to be restored), but still this can be a good compromise. For instance, for databases that contain a lot of read-only data or data that is not changed much on a day-to-day basis, the partial backup of this part of the database is a good choice because the recovery time will be low.

If large parts of the database are completely read-only, most databases allow this part of the data to be accessed in a special way, which means the database cannot modify it. This guarantees no changes to the data so that only one backup is ever required. This read-only data is common for DSS databases.

2.8.3 Physical and logical backup

Most backups are performed on the actual database files. This is either directly using AIX commands, such as `dd`, `tar`, `cpio`, and `backup`, or indirectly by RDBMS vendor tools, application vendor tools, or tools from the backup

system. This is called a *physical backup*. It is the system administrator's and database administrator's job to make sure they back up a complete set of files that make up the database.

The alternative is the *logical backup* of the database. This makes a copy of the database, but it includes the instructions on how to create the database, tables, indexes, and the table data. This copy is often in an ASCII format and in a format that is portable between different machines even with different architectures. For example, this could be used to move a database between AIX and a PC based system. Logical backups can be performed on tables or complete databases.

- In Oracle, a logical backup is performed with the Oracle `export` DBA tool.
- In DB2 UDB, a logical backup is performed with the `db2 backup` command.

The advantages are that the backup is portable between hardware platforms and operating systems and is a readable ASCII file as opposed to the normal binary database files.

The disadvantages are that a logical backup is much slower because a single tool is used to perform the backup (although some parallelization is possible for table level logical backups), and it can generate a huge file, which is much larger than the capacity of the disk; so, they are often sent directly to tape.

Logical backups are ideal for moving smaller test databases or smaller tables between systems.

2.8.4 Online and off-line backup

Everyone affiliated with database administration is comfortable with the idea of stopping the database (and any other services offered by the machine) and then, when no data can be changed, backing up the files of the system to a backup media (usually a tape). This is the classic off-line full backup and sometimes also called a *cold backup*. Once completed, that set of tapes contains the entire system that is consistent at a particular point in time.

One problem is that users are often still using the database or are connected to the database via their application that is still running late. Most sites have a policy to forcibly removing users from the system at a particular time. This can be done by forcibly halting the database; however, vendors recommend backing up the database only after a normal shutdown. To achieve this, the database is forcibly taken off-line and then restarted in a mode that stops users and then cleanly shuts down before the backup.

However, an increasing number of systems are required to be available 24 hours a day, seven days a week. This does not leave any time for a full off-line backup. One way to nearly achieve this is stopping the database, breaking a mirror copy of the database off, and restarting the database. This can reduce the database down time to a few minutes. But, even this is not acceptable to many truly 24 hours a day, global company's system and, for example, Web sites where users are online every hour of the day.

The only option is to back up with the database still running. This is called an *online backup* or *hot backup*. Most sites know the usage patterns of the database system based on user workloads. This means the backup can take place during the time of least workload. This makes the backup faster, and even though the backup may slow the system down, less users are effected, and the machine should have some spare capacity.

The various types of files on the system are treated differently.

- The AIX operating system should be backed up via the `mksysb` command.
- The user files, RDBMS configuration files, RDBMS code or application code disk can be backed up using AIX backup commands.
- The temporary or sort area files do not need to be backed up, as these are re-initialized every time the database is restarted.
- The data and index files of the database must clearly be backed up. To achieve the backing up of these files while they are being modified is impossible. So, the RDBMS vendors have special features to make this possible. The procedure is:

  1. The DBA informs the RDBMS that a particular part of the database needs to be backed up.
  2. The RDBMS stops modifying these files but puts all the updates that would go into them into the database log instead.
  3. The DBA does the backup of the data and the index files.
  4. The DBA informs the RDBMS that the backup is finished.
  5. The RDBMS searches the log for the updates and brings the files back up-to-date.

  While the backup is taking place, a lot of extra data is sent to the database log; so, there is a performance implication of online backups. Once finished, it may take some time for the RDBMS to get the data and index files updated. This also takes additional space in the log file. Note that not all of the data and indexes have to be backed up at one time. The data and

index files of one tablespace can be backed up as a group. This limits the reduction in performance, but the backup may take a little longer.

- RDBMS log disks are the final part of the database to get backed up. As the database is running, these files are being updated nearly all the time. This means that they cannot be backed up. But, an RDBMS does not just have one log. The various RDBMSs organize the logs in different ways, but they all allow the DBA to switch between log files or switch to a new file. This means that the old log file is no longer in use and can be backed up. This process is called archiving the log files. Because the log is vital for recovery, this log switching is going on regularly. To back up the RDBMS log, the DBA forces a log switch and then backs up the original log file.

Provided the DBA can find some time in the day that is less busy, then the online backup should not slow the performance of a database too badly. The problem with online backups is that you do not end up with one set of tapes that are a complete and consistent backup at one point in time. The various parts of the backup all happen at different times, and it is the database log that allows the database to recover fully. The RDBMS stores the backup files and times in the control files; so, it knows when each backup took place and the particular log files that are required to recover the database from a particular backup file. Some people find this worrying because it is not in their control.

2.8.5 Backup recommendations

- Plan for backing up your database during the design of the system for high performance. If it is added at the last minute, you may find it cannot be backed up in the available time or budget.
- Have more than one tape drive so that it does not become a single point of failure.
- Test the backup and tapes regularly.
- Perform a disaster recovery test once a year.
- Do not be afraid of online backups, as they are used often.
- The RDBMS vendor's manuals have all the backup methods and options, but they are not very good at recommending any particular strategy, schedule, or method.

There are excellent books on backing up Oracle that cover all the options in great detail, for example, *Oracle Backup and Recovery Handbook*, ISBN

0-0788-2106-1. If you want to avoid making typical mistakes and save a lot of time testing, then these books are definitely worth reading.

Chapter 3. Types of workload

In order to understand the performance issues associated with a particular database, it is helpful to have an understanding of the different database profiles and their unique workload characteristics.

RDBMS systems can be put to many different uses, that is, holding different types of data and allowing different types of processing to be performed against that data. In this chapter, we outline typical databases and their characteristic workloads, but there will always be deviations that do not fit within the models described here.

Throughout this chapter, we will use as our reference database one that contains 50 GB of raw data. Depending on the exact nature of the database, more or less disk space may be required in order to actually implement the database. We also describe typical user numbers and transaction response times for the different workloads.

Most IT departments are responsible for managing more than one database system; so, real environments are more complex than suggested here. There are typically multiple OLTP and DSS systems in use that, together, satisfy all the processing requirements of the company. Data is often moved between these databases so that different applications with different transaction types and user populations can access the data.

3.1 Online Transaction Processing (OLTP)

Online Transaction Processing (OLTP) databases are among the most mission-critical and widely deployed of any of the database types. Literally, millions of transactions encompassing billions of dollars are processed on OLTP systems around the world on a daily basis. The primary defining characteristic of OLTP systems is that the transactions are processed in real-time or *online* and often require immediate response back to the user. Examples would be:

- A point of sale terminal in a retail setting
- An Automated Teller Machine (ATM) used for withdrawing funds from a bank
- A telephone sales order processing site looking up inventories and taking order details

From a workload perspective, OLTP databases typically:

- Process a large number of concurrent user sessions
- Process a large number of transactions using simple SQL statements
- Process a single database row at a time
- Are expected to complete transactions in seconds, not minutes or hours

OLTP systems process the day-to-day operational data of a business and, therefore, have strict user response and availability requirements. They also have very high throughput requirements and are characterized by large amounts of database inserts and updates. They typically serve hundreds, if not thousands, of concurrent users, which can severely impact system performance.

Special consideration should be given to the following areas when designing an OLTP system for performance:

- Use indexes to improve performance. However, too many indexes can degrade the performance of insert and update operations.
- SQL statements should be as well-tuned as possible.
- Database block sizes should be small, in the range of 2-4 KB.

In comparison to our reference 50 GB database, a typical OLTP system would have the following characteristics:

- Would require three times the disk space - standard rule of thumb for the data plus indexes and temporary work areas.
- Would support 200 to 600 users.
- Response times would be less than two seconds.
- Specific queries might be allowed to take 45 seconds.
- Available during normal business hours, typically 8 am to 6 pm. An increasing number of systems require 24 hour, 7 day a week availability, especially international companies.
- Upgrades are planned in advance to occur over a weekend.
- The application is often written in-house but is increasingly likely to be a third party package possibly modified for particular business needs.

3.2 Online Analytical Processing (OLAP)

Modern Online Analytical Processing (OLAP) databases are based on a set of 12 rules developed by E.F. Codd in 1993 in his white-paper entitled *Providing OLAP (On-line Analytical Processing) to User-Analysts: An IT*

Mandate. This whitepaper outlined a methodology for designing systems that would be capable of providing live, ad-hoc data access and analysis.

The primary defining characteristic of OLAP databases is that they provide multi-dimensional or aggregated views of the data. The multi-dimensional data is called by many a *data cube*. This term is used to express the idea that the data has been transformed into different dimensions, such as geography, sales figures, and product line. There are many other dimensions that can be used at the same time, but it is easier to think in terms of only three dimensions.

Using an OLAP system, a user can asks such questions as: Which geographical area has the best sales? They could then select a particular geography, turn the data cube to see the next dimension (product lines), and ask: Which product lines sold best in these geographies? Then they might turn the data cube again and ask: How has that changed with time?

The aggregate views used by OLAP databases are summary tables that are used to perform these types of analyses. The RDBMS could search the entire database to answer each question, but this would be very expensive in disk I/O and CPU terms and reduce the response time or numbers of users that the system could support. After the OLAP database has received more data from other systems, the database creates a series of summary tables containing averages and totals for sales figures from the individual sales details. The application then uses these much smaller summary tables to answer most of the user and application queries.

The workload characteristics of OLAP databases differ from those of OLTP databases in that they:

- Serve a smaller number of users
- Support a large number of queries using complex SQL statements
- Process many database rows at a time

In comparison to our reference 50 GB database, a typical OLAP system would have the following characteristics:

- Would require five times the amount of disk space to allow for the large summary tables.
- Would support 20 to 50 users.
- Response times would be less than 20 seconds.
- Large user requests that require the query to read the fine details from the database might be allowed to take 15 minutes.

- Available during normal business hours. The database needs to take data updates from OLTP systems so that the aggregrates can be re-created or updated, usually every night.
- The vast bulk of the database is very static, only the last month's data changes.
- System down time allows for improvements of the summary tables based on the results of monitoring user activity.
- The application is nearly always a third party package with a business model created for this particular company's product or services.

3.3 Decision Support Systems (DSS)

Decision Support Systems (DSS) differ from the typical transaction-oriented systems in that they most often consist of data extracted from multiple source systems for the purpose of supporting end-user:

- Data analysis applications using pre-defined queries
- Application generated queries
- Ad-hoc user queries
- Reporting requirements

DSS systems typically deal with substantially larger volumes of data than OLTP systems due to their role in supplying users with large amounts of historical data. Whereas 100 gigabytes would be considered large for an OLTP system, a large DSS system would most likely be 1 terabyte or more. The increased storage requirements of DSS systems can also be attributed to the fact that they often contain multiple, aggregrated views of the same data.

While OLTP queries tend to be centered around one specific business function, DSS queries are often substantially more complex. The need to process large amounts data results in many CPU intensive database sort and join operations. The complexity and variability of these types of queries must be given special consideration when designing a DSS system for performance.

DSS systems fall into two categories: Data warehouse and data mart. Both data warehouses and data marts have the following workload characteristics:

- Complex and very complex SQL statements
- Long running SQL queries that may take minutes or hours to complete
- Diverse query types that answer complex business questions

- Applications often model the logical structure of the business and hide the database and SQL structure from the end-user
- Perform mostly full table scans or use summary tables built by the database administrator

3.3.1 Data warehouse

Data warehouses normally consist of a single large server that serves as a consolidation point for enterprise data from several diverse database systems. Data warehouses typically contain years worth of historical data in order to serve as a single source of information for all the decision support processing in the entire business organization. Whereas OLTP systems are primarily concerned with changing the data contained in the database, data warehouses are used to extract data for end user reporting and data analysis needs.

In comparison to our reference 50 GB database, a typical data warehouse would have the following characteristics:

- The data warehouse might include six other databases and contain a great deal of historical information, therefore, increasing the database size by a factor of 10 (500 GB) and increasing the disk space required by a factor of 3 (1500 GB).
- Limited number of users with *super-user* authority that create the data aggregates and perform cleanup operations on the data.
- Response times could be one hour to one week.
- System available 24 hours a day with particular tables taken off-line for periodic updates.
- Supports one view of the business data extracted to data marts for further analysis.
- The application is often written in-house but is increasing likely to be a third party package to automate common and repetitive work.

3.3.2 Data mart

Data marts can be defined as more narrowly focused data warehouses. Data marts are created with a subset of the operational production data in order to satisfy the reporting needs of a specific organizational unit or to solve a particular business problem.

In comparison to our reference 50 GB database, a typical data mart would have the following characteristics:

- Would require five times the amount of disk space - standard rule of thumb for building the data mart. Extra space would be required for new data to be cleaned and modified before adding to the database and extra summary tables
- Would support 20 to 100 users.
- Response times would be less than two minutes, many simpler requests under 10 seconds. Once the data is extracted to the user tool, the users analyze the data for five to 30 minutes before requesting more.
- Large user requests might be allowed to take 20 minutes.
- Available during normal business hours.
- System taken down for data loads and creation of summary tables.
- Data can be reextracted and loaded from the warehouse or original data source.
- Application third party package resides on PC for graphical modeling and analysis. Often there is a business model on the PC to hide the complexities of the database design and SQL.

3.3.3 Business Intelligence (BI)

Business Intelligence (BI) is a general term used to describe the process of extracting previously unknown, comprehensible, and actionable information from large databases and using that information to make *intelligent* business decisions.

3.3.4 Data mining

Data mining is a relatively new data analysis technique used to find and exploit previously unknown relationships between seemingly unrelated data. Unlike traditional reporting and multi-dimensional analysis techniques in which very specific queries are used to extract the data, data mining uses several statistical data analysis algorithms to extract previously unknown information from the existing data.

For example, a data mining application might study car insurance claims to spot particularly good or poor risk customers, and their insurance premiums could change as a result. The application can spot correlations in the data, such as particular areas of the country, age groups, car types, and jobs that might otherwise go undiscovered.

Unlike OLAP and Data Marts, where the user formulates a question and the RDBMS finds the answer (or graphs the data), which might then pose further

questions, the data mining application not only finds the answers but goes on to explain the relationships between the selection criteria upon which the answers are based. Care must be taken to ensure the new information is really a trend and not a quirk in the data.

Data mining solutions often consist of a mixture of database tools and technologies. These include such IBM products as DB2 OLAP Server and Intelligent Data Miner.

3.4 Enterprise Resource Planning (ERP)

Enterprise Resource Planning (ERP) systems have gained enormous market share in the last decade primarily due to their ability to consolidate multiple data sources into one system offering tightly integrated applications. They are typically implemented in a *three tier* client/server configuration with the database at the core of the system. Application vendors, such as SAP, Baan, JBA, PeopleSoft, Siebel, Retek, and Oracle Financials are among the current market leaders in the ERP arena.

ERP systems can be considered something of a hybrid from a database workload perspective, as they most often exhibit the same workload characteristics as traditional OLTP, OLAP, DSS, and batch reporting systems. Originally, ERP systems were primarily used in an OLTP capacity, but increasingly, they are offering more DSS functions or modules as an extension.

ERP systems have the following characteristics:

- They are three tier client/server solutions in that they include:
  1. A central database server.
  2. Multiple application servers - Are mainly compute bound systems running a few processes that actually do the work for users and cache data from the database server for performance. A typical configuration would be one database server with four to eight application servers. These application servers also limit the number of user connections to the database.
  3. User workstations (normally PC based) that provide user friendly GUI based applications and graphics.
- The databases structures are complex having hundreds or thousands of tables.
- The application is completely generic. As supplied, it needs a lot of tailoring to meet the specific business needs of a company. Most of this

tailoring goes into the tables of the database. For example, the various divisions, departments, and reporting structures and their cost and profit centers would be stored in the database tables. The effect of this on the database is that queries are much more complex than those on OLTP systems and join many more tables.

- To increase response times and the number of users supported, most ERP vendors do not allow users to directly modify the database. The updates are typically queued to background processes that do the updates at a later time. This reduces database locking problems and speeds up user response times.
- Serve a large number of users (same as OLTP systems).
- Process a large number of transactions utilizing both simple and complex SQL statements (mixture of OLTP and OLAP).
- Interactive user response times are measured in seconds (same as OLTP).
- Process many database rows at a time (same as OLAP).

Workloads should be clearly defined and separated in order to avoid performance problems associated with mixing OLTP, OLAP, DSS, or batch reporting activities on the same system at the same time.

In comparison to our reference 50 GB database, a typical ERP system would have the following characteristics:

- Would require three to five times the disk space (standard rule of thumb).
- Would support 100 to 1000 users.
- Response times would be less than four seconds.
- Available 24 hours, 7 days a week for most production systems. Development and test systems would need to be available during normal business hours.
- Zero system down-time for production systems; 4 to 8 hours per night for development and test systems due to the lack of a 24 x 7 availability requirement.
- The application would be a well known third party package possibly modified for particular business needs during the implementation phase, which is likely to take many months.

3.5 e-Business

e-business, as defined by IBM, is *any activity that connects critical business systems directly to their critical constituencies (customers, employees, vendors, and suppliers) via intranets, extranets, and over the World Wide Web*. The explosive growth of the Internet in the mid 90s has forced businesses to radically change the way in which services are provided to their customers. Companies have had to Web-enable core business processes to strengthen customer service operations, streamline supply chains, and reach new and existing customers. These changes have forever altered customer's expectations regarding support and response.

At the core of any e-business transaction is the database used to satisfy user queries. These queries are typically generated as a result of user input via HTML pages that eventually end up being processed by the database server. The Web server normally has a special interface module that calls the database server to supply the information. The Web server also has to provide the information back to the user in HTML format so that the raw database data has to be packaged up before being returned to the Web server. The generation of the connection to the database as a result of the user request is known as a *Web hit*.

Connecting and disconnecting to a database takes a lot of processing power and is not a good option for each Web hit in terms of providing good performance. Therefore, a *Web application server* is typically used to provide this permanent connection and sits between the Web server and the database server.

When compared to an OLTP or ERP database, a Web database is actually small in size. The actual data content is not very large for most Web sites; however, there are exceptions. A less desirable alternative is to connect the Web server and application server to the real production OLTP or ERP system. The major risk is security, as a Web hacker effectively has a network connection to your vital database. This is one reason why there is such a great deal of interest in security on the Web and why there are a multitude of products to help provide this security.

In workload terms, these databases are like OLTP systems, but there is an additional problem. Unlike a company machine where the user population is well known and understood, the Web is unpredictable. Sizing a Web-enabled database can be very hard to nearly impossible due to the volatility in the number of users accessing the site and the tendency to initially underestimate the demand that will be placed on the servers.

There are two different approaches that can be taken when sizing the system:

1. Over-specify the machine by a factor of two or three to ensure it can take the unexpected peak hit rates.
2. Have a machine that can be rapidly upgraded, particularly from a CPU standpoint.

Also, a high availability solution (like IBMs HACMP) should be in place from the start, as a site that is failing or very slow can quickly result in users migrating to a competitor's Web site.

In comparison to our reference 50 GB database, a typical e-business system would have the following characteristics:

- Would require three times the disk space (like OLTP).
- No real concept of a user, but each database hit usually results in the execution of a database transaction.
- Response times would be less than two seconds.
- Available 24 hours, 7 days a week due to the fact that it is connected to the Web.
- The application would be a third party package, possibly modified for style.

3.6 Reporting

It might sound strange that, in what many people thought should be a paperless world by now, many large databases are used simply to create reports. Often small volumes of reports can be performed on the customer's live OLTP system. However, the fact that report generation often involves processing huge volumes of data can cripple OLTP performance. The most common solution is to move the data to a dedicated report database system.

While some databases receive their information from the customer's live OLTP systems, others are updated directly for the purposes of data collection and summarization. An example would be a water company that receives data from thousands of water flow meters into the system and automatically updates the database at a defined interval. The database is then used to report:

- The state of the overall system
- Quirks that might need investigation
- Data used to spot long term trends

There are some third party or RDBMS tools that are often used to generate reports. These tools allow an advanced user or DBA to reformat the report and describe the data columns and totals. The user then activates the report after supplying specific limitations, such as the period in which the report should cover or particular areas to be included or excluded. An example would be the sales report for the North region for the past three months.

Often, there is a trade off between using the report tool, which is quick to implement and simple to modify and maintain, and performance. For high capacity, high volume reports customers typically use the report tool as a prototype and then recode the report in a 3 GL language, such as C or COBOL for better performance.

In comparison to our reference 50 GB database, a typical reporting system would have the following characteristics:

- Would require three times the amount of disk space - very similar to the OLTP database.
- Would support one to 40 users. Access might be via a batch queue for ad-hoc reports or a scheduling system for a report needed at fixed intervals.
- Response times are dependent on the nature of the report. Large reports, for example, could take eight hours or more.
- Available nearly 24 hours, 7 days a week.
- The system will be taken down for updates and backups only.
- The application would consist of RDBMS tools or a third party package with very specific, pre-defined reports.

Chapter 4. Specific databases

This chapter describes the different physical and logical structures for DB2 UDB and Oracle as well as the processes used to access and manipulate the databases. It is recommended that these basic concepts are totally comprehended since they are referenced in the other chapters.

4.1 DB2 UDB Database architecture

The DB2 UDB databases are stored in both physical and logical structures. When the database is accessed either locally or remotely, some internal DB2 UDB processes will interact with another structure created on memory. DB2 UDB databases consist of the following:

- Physical Structure:
  This is where the data and all objects that belong to a database are stored.
- Logical Structure:
  Logically divides the objects inside the database.
- Memory Structure:
  Holds the information that is necessary to process the requests generated by the applications and user connections.
- Processes:
  Accesses the memory structure, manipulates, and returns data requested by the applications and user connections.

This entire structure makes DB2 UDB easily configurable for the different system workload characteristics. All possible administration tasks can be done either locally on the AIX platform or remotely from a workstation.

4.1.1 Memory structures

When DB2 UDB is running on an AIX operating system, the amount of real and virtual memory used will basically depend on how the database manager (instance) parameters and database parameters are set. Based on these parameters, the RDBMS will allocate more, or maybe less, memory resources to process the requests generated by the applications and user connections.

It is important to point out that, if the resources requested by DB2 UDB cannot be satisfied with real memory, some pages will be requested from paging space, which causes a performance degradation.

The memory will be allocated for four different memory requestors. The memory allocation for each of them will happen at different times depending on the following:

- Database Manager Shared Memory: Is allocated when the command `db2start` is run and deallocated with the `db2stop` command execution.
- Database Global Memory: Is allocated when the database is activated using the command `ACTIVATE DATABASE` or when the first connection is established
- Application Global Memory: Is allocated when an application connects to a database in a partitioned environment or when the intra_parallel parameter is enabled. This memory is used by agents working on behalf of the application in order to share data and coordinate activities among themselves.
- Agent Private Memory: Is allocated when an agent is assigned to work for a particular application.

Please refer to Chapter 12, “DB2 UDB tuning” on page 259 for further explanation of which DB2 UDB parameters can be tuned for better system performance.

4.1.2 Logical storage structures

The primary data storing object in a database is a row. A row is composed of one or more columns that store logically related information. A table is composed of a certain numbers of rows, from zero to an undetermined number. The amount of existing rows in a table defines the table cardinality. Cardinality means number of distinct values.

A DB2 UDB table can store data in the following listed types of columns:

- SMALLINT - Small integer
- INTEGER - Large integer
- BIGINT - Big integer
- FLOAT - Single or double precision floating-point number
- DECIMAL - Decimal number
- CHARACTER - Fixed-length character string of length integer
- VARCHAR - Varying-length character string
- LONG VARCHAR - Varying-length character string
- BLOB - Binary large object string
- CLOB - Character large object string
- DBCLOB - Double-byte character large object string
- GRAPHIC - Fixed-length graphic string
- VARGRAPHIC - Varying-length graphic string

- LONG VARGRAPHIC - Varying-length graphic string
- DATE - Date
- TIME - Time
- TIMESTAMP - Timestamp
- DATALINK - A link to data stored outside the database
- DISTINCT-TYPE-NAME - A user-defined type that is a distinct type
- REF - Reference to a typed table

See both volumes of the *DB2 UDB SQL Reference*, SC09-2847 and SC09-2848, for details on the supported data types.

Once the database administrator defines a table with the appropriate columns, the table is ready to maintain some relationship with other tables, and their data can be retrieved though a standardized language called SQL (Structured Query Language).

In order to speed up the access to the data in a table, one or more indexes can be created. Indexes consist of search-key values and pointers to the rows containing those values.

All the tables and indexes are stored on logical space divisions called *tablespaces*. There are two types of tablespaces:

- SMS - System Managed Space. Data is stored in file system directories.
- DMS - Database Managed Space. Data is stored in file system files and/or on raw devices.

The physical disk space assigned to a tablespace is called container.

Both types of tablespaces allow the user to store tables and indexes together in the same tablespace. However, if DMS tablespaces are used, it is possible to logically split a single table into three different tablespaces containing, respectively, the data, the indexes, and the Large Objects (LOB) type columns. This operation allows the same table to be stored on three different devices, thus, enabling parallelism and performance improvement.

By default, three SMS tablespaces are created:

- SYSCATSPACE - Stores the internal database control tables called catalog tables
- USERSPACE1 - Stores the user defined tables
- TEMPSPACE1 - Used to store temporary tables for operations, such as sorts and reorganizations

Tablespaces are stored in nodegroups. Nodegroups are a set of one or more database partitions. When using the DB2 UDB Enterprise Edition, only one database partition exists on the machine. However, when using the DB2 UDB Enterprise-Extended Edition, one or more database partitions can be defined on a system.

All the tables, indexes, catalog tables, DB2 UDB objects, tablespaces, and nodegroups form an entity called a database. Each database has its own set of physical control files, log files, and data files. For more information about the physical structure please refer to 4.1.3, “Physical storage structures” on page 59.

One or more existing databases reside within an instance. An instance is a complete environment that holds the databases. It controls the access to all databases created within it. Other instances access those databases by communicating with the owning instance. The more instances defined on a system, the more machine resources will be needed.

Each physical machine represents one system that is composed of one or more instances. Figure 12 illustrates how the DB2 UDB is logically structured.

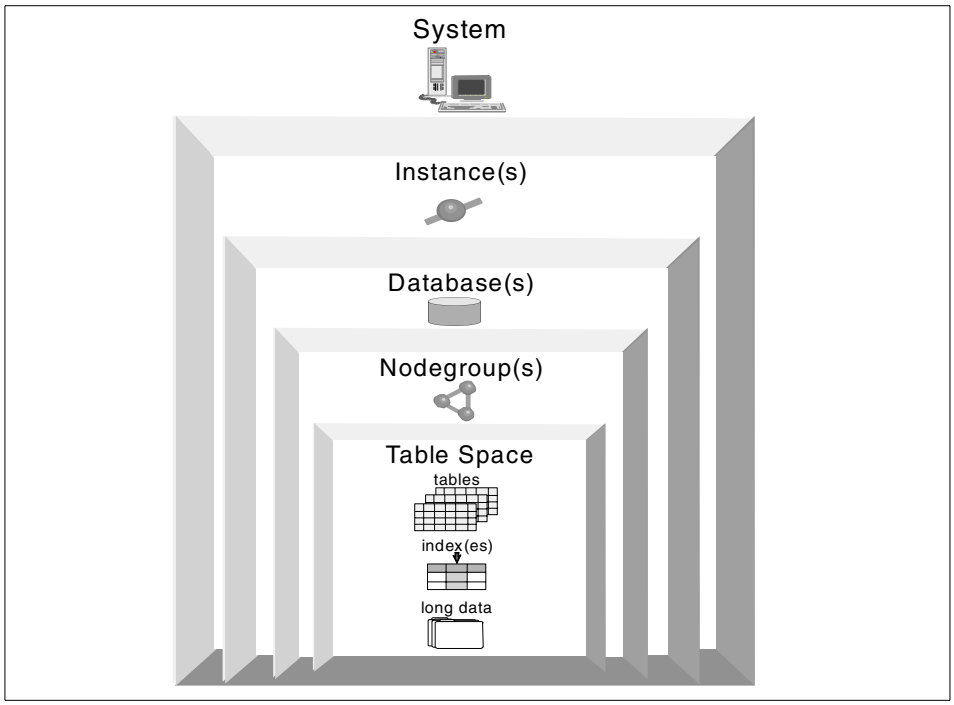


Figure 12. DB2 UDB logical structure

4.1.3 Physical storage structures

When the DB2 UDB Version 6.1 code is installed on an RS/6000 machine, all the product files will be located in the directory /usr/lpp/db2_06_01.

Each instance will be created on a physical directory that is the home directory of the userid defined at the instance creation time. This userid will have the same name as the instance name and will have the system administrator privilege for this instance.

After the instance is created and started, databases can be created within it. Databases are created on existing directories or file systems specified by the `ON` clause on the `create database` command:

```
create database sample on /db_sample
```

This command will create the database sample on the following physical structure:

/db_sample/instance_name/NODE0000/SQL00001

where:

instance_name is the name of the instance where the database was created.

NODE0000 identifies the partition number in a partition environment. For a machine where DB2 UDB Enterprise Edition is running, this entry is always NODE0000.

SQL00001 identifies the sequence in which the databases were created. The next database that is created will be located under SQL00002.

4.1.3.1 Internal files

After the database is created, some internal control files will be located under the database directory /db_sample/instance_name/NODE0000/SQL00001. Each file will be responsible for a different database functionality, as follows:

- SQLDBCON - Stores the database configuration parameters and flags for the database
- SQLOGCTL.LFH - Tracks and controls all of the database's log files
- Syyyyyyy.LOG - Restores the database into a consistent state in the event of a database failure situation or system crash
- SQLINSLK and SQLTMPLK- Ensures that a database is only used by one instance of the database manager
- SQLSPCS.1 - Contains the definition and current state of all table spaces in the database.

- SQLSPCS.2 - Is a copy of SQLSPCS.1
- SQLBP.1 - Contains the definition of all the buffer pools used in the database
- SQLBP.2 - Is a copy of SQLBP.1
- DB2RHIST.ASC - Is the database recovery history file
- DB2RHIST.BAK - Is a copy of DB2RHIST.ASC

These files are used exclusively by the RDBMS and should not be deleted or removed from this location.

4.1.3.2 Log files

The log files, Syyyyyyy.LOG, will have their file names varying from S0000001.LOG through S9999999.LOG. This files are used to ensure the database's consistency in case a failure or if a system crash occurs. The number of existing logs in a database will depend upon the log size, number of primary logs defined, number of secondary logs defined, the setting of the LOGRETAIN parameter, and the amount of transactions that insert, update, and delete data rows.

The size of the logs used by the database (measured in 4 K pages) will be determined by the database configuration parameter logfilsiz. The number of primary and secondary logs will be determined by the database parameters logprimary and logsecond, respectively.

By default, the database uses circular logging, that is, the connections will record their transactions on the primary logs until they are filled. At this time, the secondary log will be allocated according to the amount of log space requested. The logging operation continues until both the primary and secondary logs are completely used. From this point on, each command that generates a change to be recorded on the log files will be refused unless a `COMMIT` or `ROLLBACK` statement is executed. This logging behavior is ideal for a read-only environment, when few database changes are expected, since the number of primary and secondary logs is limited.

However, if the database is updated frequently, the recommended logging mode is the log retention logging. With this approach, a command will only be rolled back when its unit of work exceeds the total amount of space defined by the sum of primary and secondary logs. The number of log files will increase until they reach S9999999.log, then the log name counter will be restarted. Using this logging method also allows an online database backup, a backup of chosen tablespaces, as well as a point-in-time recovery.

Please refer to 7.6.1, “DB2 UDB backup/restore scenario” on page 133 for more information about database and tablespace recovering scenarios.

4.1.3.3 Data files, index files, and temporary space

Table rows and indexes are stored in tablespaces, as well as temporary tables that are used for reorganizations and sorts. The tablespaces can be either System Managed Space (SMS) or Database Managed Space (DMS). The main difference is how the disk space will be allocated and how these tablespaces will increase in size. The physical disk space allocated for a tablespace is called container. Containers store data and indexes in disk pages of 4 k, 8 k, 16 k, or 32 k sizes. The choice for the value to be used will depend basically on the system workload as well as on SQL limits. See also Appendix A "SQL Limits" in the *IBM DB2 UDB SQL Reference, Volume 2*, SC09-2848.

SMS tablespaces

By default, three System Managed Space (SMS) tablespaces are created when a database is created. The physical storage space allocation and management tasks within the SMS will be the responsibilities of the operating system's file system manager. The data and index pages will be recorded in files within file systems or directories. Every time the amount of data, indexes, or temporary tables increases, DB2 UDB will determine the name of the files to extend in order to accommodate this data, and the operating system will be responsible for managing them. The two most important characteristics of this tablespace type is that the disk space is not pre-allocated and, once the number of containers is specified, it can not easily be changed. Only through an operation called *redirected restore* it is possible to add more containers to an SMS tablespace.

The following is the list of files that compose an SMS tablespace:

- SQLTAG.NAM - One for each container subdirectory and used to verify that the database is complete and consistent
- SQLxxxxx.DAT - Table file, where all rows of a table are stored, with the exception of LONG VARCHAR, LONG VARGRAPHIC, CLOB, BLOB, and DBCLOB data
- SQLxxxxx.LF - File containing LONG VARCHAR or LONG VARGRAPHIC data
- SQLxxxxx.LB - Files containing BLOB, CLOB, or DBCLOB data
- SQLxxxxx.LBA - Files containing allocation and free space information about the SQLxxxxx.LB files
- SQLxxxxx.INX - Index file for a table
- SQLxxxxx.DTR - Temporary data file for a REORG of a DAT file
- SQLxxxxx.LFR - Temporary data file for a REORG of a LF file
- SQLxxxxx.RLB - Temporary data file for a REORG of a LB file
- SQLxxxxx.RBA - Temporary data file for a REORG of a LBA file

DMS tablespaces
When Database Managed Space (DMS) tablespaces are used, the database manager will be responsible to control the physical storage space allocation and management.

The containers for a DMS tablespace are raw devices or file system files with pre-defined sizes, that is, the disk space is pre-allocated when a DMS tablespace is defined. Usually, a DMS tablespace performs better than an SMS since it does not have to spend time extending a lot of files when new rows are inserted. Using DMS tablespaces also allows a single table to store its data, index, and large objects on up to three different DMS tablespaces, thus, improving performance through parallel disk I/O. Also, a DMS tablespace can be easily increased by just adding new containers to it.

4.1.4 Processes

The DB2 UDB RDBMS architecture is based on processes. Through this architecture, DB2 UDB is able to communicate with remote and local client applications.

For each client application that connects to a database, a single coordinator agent is assigned. This coordinator agent is a process named *db2agent*, and it is responsible for serving the requests from the clients to the database. On a database where the intra-partition parallelism feature is enabled, the db2agent will use one or more agent processes, named *db2agntp*, and coordinate their work in order to retrieve the information requested by the clients more quickly. Please refer to 5.2.3, “Inter-partition and intra-partition parallelism” on page 86 for more information about the different types of parallelism.

As client applications could easily interfere with the internal engine processes, the DB2 UDB implements a firewall that isolates the DB2 UDB engine's processes from the application processes in order to avoid a possible instance crash. Apart from that, two other processes remain outside the firewall:

- db2udfp- Fenced user-defined functions (UDFs)
- db2dari- Fenced stored procedures

The following figure illustrates how the DB2 UDB is structured:

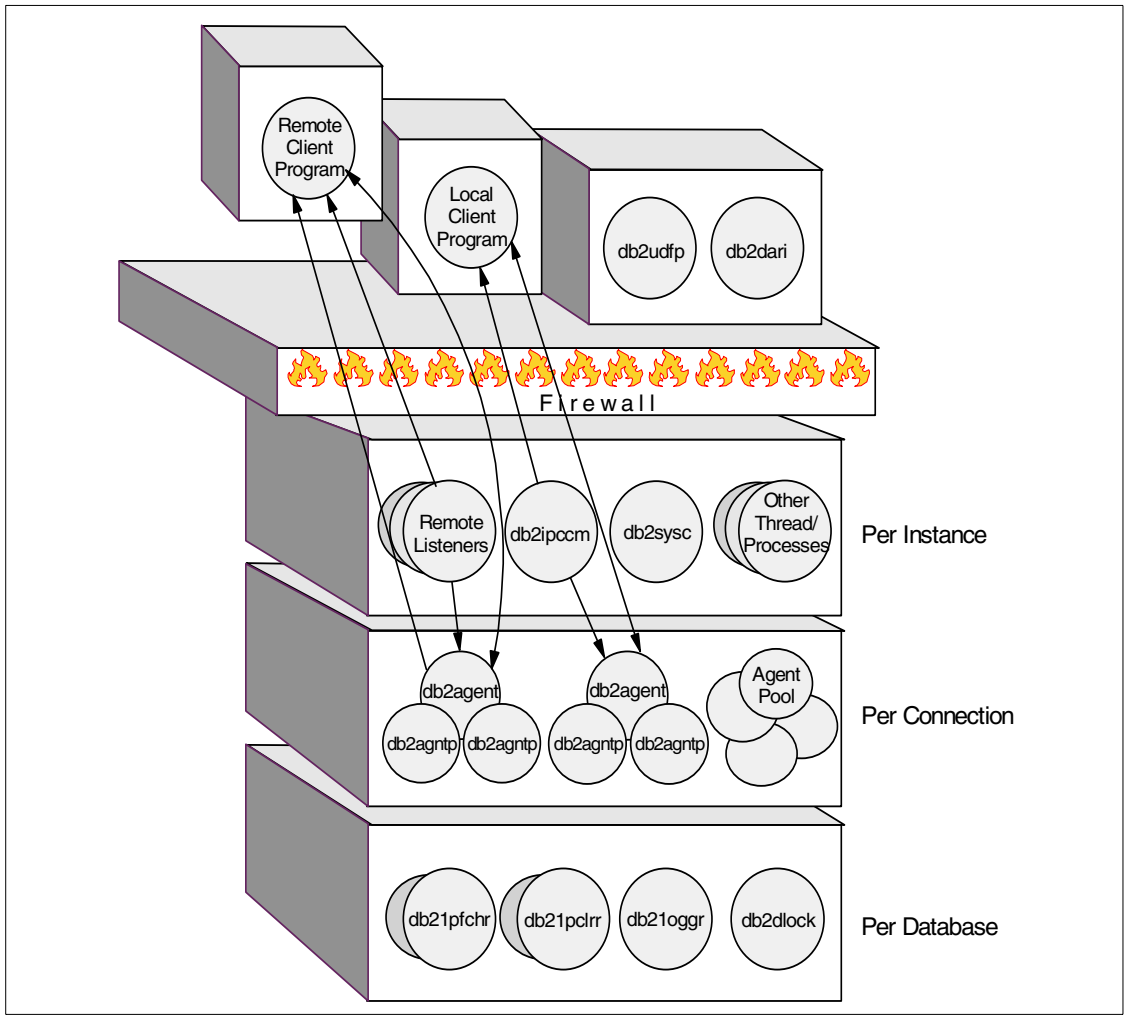


Figure 13. DB2 process model

The DB2 UDB listeners are responsible for the communication between the client and server processes when the clients ask the server for a connection. For each different communication protocol, there is an associated listener. They will only be available if DB2 UDB is asked to support a specific communication protocol (defined at the DB2COMM registry variable). For treating local connections, a special inter-process communications (IPC) listener will be used. The IPC processes can be queried with the `ipcs` command.

There can be up to four listeners available in an AIX environment:

- db2ipccm - For local client connections
- db2tcpcm - For TCP/IP connections
- db2snacm - For APPC connections
- db2tcpdm - For TCP/IP discovery tool requests

Each different instance on an AIX machine will have their own set of processes that are responsible to keep the instance running as well as managing other operational needs. Basically, this system availability is achieved through the use of the system controller process *db2sysc*.

Along with db2sysc, the following processes are also needed at the instance level:

- db2resyn - The resync agent that scans the global resync list
- db2gds - The global daemon spawner on UNIX-based systems that starts new processes
- db2wdog - The watchdog on UNIX-based systems that handles abnormal terminations
- db2fcmdm - The fast communications manager daemon that handles inter-nodal communication (used only in DB2 EEE)
- db2pdbc - The parallel system controller, which handles parallel requests from remote nodes (used only in DB2 EEE)
- db2panic
  the panic agent, which handles urgent requests after agent limits have been reached at a particular node (used only in DB2 EEE)

DB2 UDB also uses the following processes for each created database:

- db2pfchr - For input and output (I/O) prefetching
- db2pclnr - For buffer pool page cleaners
- db2loggr - For manipulating log files to handle transaction processing and recovery
- db2dlock - For deadlock detection

Among all the possible client, listener, database, and instance processes, the db2agent (or db2agntp on an intra parallel database) is the one that can allocate and possibly hold the largest amount of memory resource in the system.

When a connected application ends the connection, the agent (or agents) associated with that specific application turn to an idle status. The number of

agents allowed to remain idle, but still holding resources, will be determined by the database parameter *num_poolagents*.

In order to verify all the DB2 UDB processes running on a AIX machine, the command `ps -ef | grep db2` can be issued.

None of the DB2 UDB processes should be killed by the system or database administrator since any attempt in this direction could cause the whole instance to crash. DB2 UDB controls the creation and removal of all the existing processes from memory.

4.1.5 SQL extensions - Stored procedures

Many programmers code repetitive sequences of inserts, updates, deletes, and selects. Depending on how often the same sequence is executed by the client machines, the usage of stored procedures might be recommended. Stored procedures are programs that reside on the server machine and that can be called by the client machine through a regular `CALL` command within a transaction.

The usage of stored procedures can improve the overall database performance by reducing the number of the transmissions on the network. The faster the network can send a request and deliver the output, the quicker the client application will finish processing.

The stored procedures can be written in four different languages: JAVA, SQL Procedure, Cobol, and C.

The choice of which language should be used will depend on how familiar the application designers are with the language. In the other cases, it is recommended that SQL Procedure language or JAVA are used due to their easy writable code.

- JAVA

  DB2 UDB Stored Procedure Builder Tool generates stored procedures written in JAVA without the need to know the language. Only the SQL statements need to be defined. DB2 UDB also implements support for both static and dynamic SQL in the body of the stored procedure.

- SQL Procedure

  SQL Procedure is an IBM programming language extension to the SQL language based on the ANSI/ISO standard language SQL/PSM. It is similar to Sybase, Microsoft SQL, Oracle, and Informix SQL languages, which allows the users of those RDBMSs to easily get adapted to it.

4.1.6 Administration tools

The DB2 UDB Administration Client provides all tools necessary to administer a server. The main tool for database administration is the *Control Center.*

Control Center is a GUI tool that provides a clear overview of all the objects within a DB2 UDB system. Through this graphical tool, it is possible to create objects, such as databases, tablespaces, bufferpools, tables, indexes, and triggers.

It can be launched through the command `db2cc` and looks as follows:

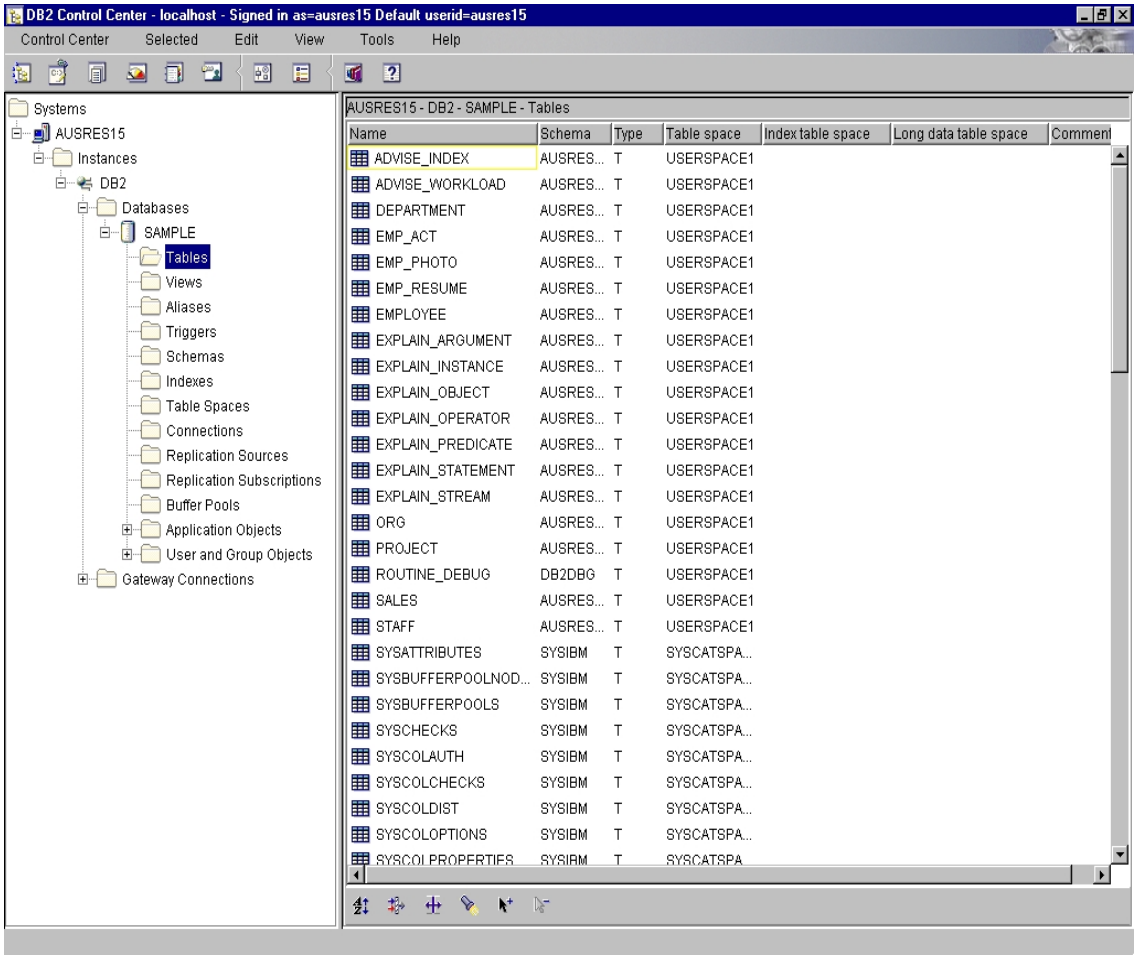


Figure 14. DB2 UDB Control Center

Besides enabling easy object creation and administration, it also allows to launch other administration tools provided by DB2 UDB:

- Script Center
  The Script Center enables you to create, run, and schedule operating-system-level commands and DB2 command scripts.
- Alert Center
  The Alert Center notifies you when thresholds that you have set have been exceeded or when a node in a multi-node environment is no longer responding.
- Journal
  The Journal allows you to view the status of jobs and to view the recovery history log and messages log.
- Information Center
  The Information Center gives you quick access to the information in the DB2 product manuals and sample programs and provides access to other sources of DB2 information on the Web.
- License Center
  The License Center displays the status of your license as well as it allows you to configure your system for proper license monitoring.

For database administration beginners, DB2 UDB provides SmartGuides. SmartGuides are GUI windows that help the database administrators to understand more clearly the tasks that they are performing through a step-by-step panel that, based on the information received, is able to create and execute commands and also recommend changes in order to increase the database performance.

The SmartGuides are available for the following tasks:

- Back up database
- Create database
- Create table
- Create table space
- Index SmartGuide
- Performance configuration
- Restore database
- Configure multi-site update SmartGuide

There are other GUI tools that are not available directly from the Control Center toolbar but have the same administration importance as the tools available on the Control Center. The following is a list of these tools and a brief description of their tasks:

- Performance Monitor
  Tool used to monitor DB2 UDB objects such as databases, tables, and tablespaces, thus, allowing the database administrator to easily tune the database.

- Event Monitor
  Collects monitoring information for a specified database activity during a period of time.
- Event Analyzer
  Used to analyze the output generated by the Event Monitor.
- Visual explain
  Allows the database administrator to investigate how DB2 UDB accesses and retrieves the data.
- Client Configuration Assistant
  Configures access to other servers by cataloging them on the current system.

4.2 Oracle database architecture

Oracle databases have both a logical and physical storage structure. This separation of physical from logical allows the physical aspects of the database to be managed without affecting access to the logical storage structures. The logical storage structures dictate how the actual physical space of a database is used.

Oracle also uses several memory and process structures to access and manage the database. The memory structures are used to hold executing program code as well as the actual data from the underlying physical storage structures. The various Oracle processes perform such tasks as interfacing with user application programs and monitoring the database for availability and performance.

4.2.1 Memory structures

Oracle utilizes several different types of memory structures for storing and retrieving data in the system. These include the *System Global Area (SGA)* and *Program Global Areas (PGA).* These memory structures and the relationship between them is depicted in Figure 15 on page 69.

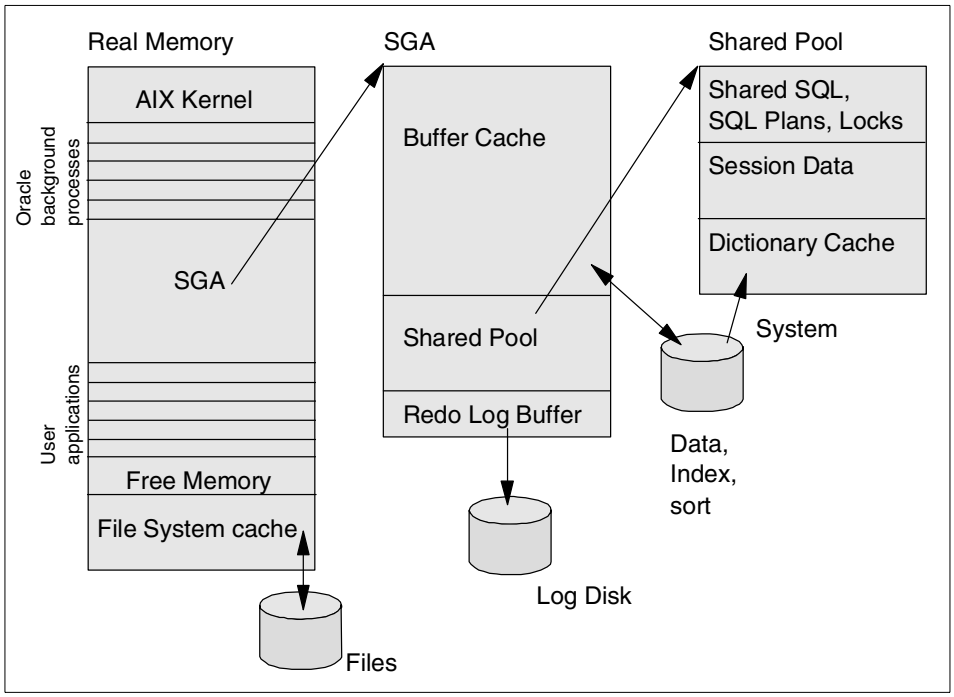


Figure 15. Oracle memory structures

The Oracle SGA is a shared memory region used to hold data and internal control structures of the database. This shared memory region details can be displayed in AIX with the `ipcs -ma` command. The Oracle SGA and its associated background processes, described in Section 4.2.4, “Processes” on page 73, are known as an Oracle *instance*. The instance identity is called by the short name *SID*. This short name is used in all Oracle processes connected to this instance. To connect to a particular instance, set the shell variable $ORACLE_SID to the SID. The SGA memory region is allocated upon instance startup and de-allocated when the instance is shut down and is unique to each database instance. The information contained in the SGA is logically separated into three different areas: The *database buffer cache,* the *redo log buffer,* and the *shared pool*.

The database buffer cache consists of Oracle database data blocks or *buffers* that have been read from disk and placed into memory. These buffers are classified as either *free, dirty,* or *pinned.* Free buffers are those that have not yet been modified and are available for use. Dirty buffers contain data that has been modified but has not yet been written to disk. Lastly, pinned buffers are buffers that are currently being accessed.

Oracle uses a Least Recently Used (LRU) algorithm to age the dirty data blocks from memory to disk. A LRU list is used to keep track of all available buffers and their state (dirty, free, or pinned). Infrequently accessed data is moved to the end of the LRU list where it will eventually be written to disk by the Oracle DBW*n* process (see Section 4.2.4, “Processes” on page 73) should an additional free buffer be requested but not available.

The size of the database buffer cache is determined by a combination of the Oracle initialization parameters *DB_BLOCK_BUFFERS* and *DB_BLOCK_SIZE*. Oracle allocates one Oracle data block of size *DB_BLOCK_SIZE* for each buffer specified by *DB_BLOCK_BUFFERS.*

The redo log buffer is used to store information about changes made to data in the database. This information can be used to reapply or *redo* the changes made to the database should a database recovery become necessary. The entries in the redo log buffer are written to the online redo logs by the LGWR process (see Section 4.2.4, “Processes” on page 73). The size of the redo log buffer is determined by the *LOG_BUFFER* parameter in the Oracle initialization file.

The shared pool area stores memory structures, such as the shared SQL areas, private SQL areas, and the data dictionary cache. Shared SQL areas contain the parse tree and execution plan for SQL statements. Identical SQL statements share execution plans. One memory region can be shared for multiple identical Data Manipulation Language (DML) statements, thus, saving memory. DML statements are SQL statements that are used to query and manipulate data stored in the database, such as `SELECT`, `UPDATE`, `INSERT`, and `DELETE`.

Private SQL areas contain Oracle bind information and runtime buffers. The bind information contains the actual data values of user variables contained in the SQL query.

The data dictionary cache is used to hold information pertaining to the Oracle data dictionary. The Oracle data dictionary serves as a roadmap to the structure and layout of the database. The information contained in the data dictionary is used during Oracle’s parsing of SQL statements.

4.2.2 Logical storage structures

An Oracle database is made up of several logical storage structures including: *Data block*s, *extents* and *segments*, *tablespaces,* and *schema objects*.

The actual physical storage space in the datafiles is logically allocated and deallocated in the form of Oracle data blocks. Data blocks are the smallest unit of I/O in an Oracle database. Oracle reserves a portion of each block for maintaining information, such as the address of all the rows contained in the block and the type of information stored in the block. This overhead is normally in the range of 84 to 107 bytes.

An extent is a collection of contiguous data blocks. A table is comprised of one or more extents. The very first extent of a table is known as the *initial extent*. When the data blocks of the initial extent become full, Oracle allocates an *incremental extent.* The incremental extent does not have to be the same size (in bytes) as the initial extent.

A segment is the collection of extents that contain all of the data for a particular logical storage structure in a tablespace, such as a table or index. There are four different types of segments, each corresponding to a specific logical storage structure type:

- Data segments
- Index segments
- Rollback segments
- Temporary segments

Data segments store all the data contained in a table. Likewise, index segments store all the data contained in an index. Rollback segments are used to hold the previous contents of an Oracle data block prior to any change made by a particular transaction. If any part of the transaction should not complete successfully, the information contained in the rollback segments is used to restore the data to its previous state.

Rollback segments are also used to provide *read-consistency*. There are two different types of read-consistency: *Statement-leve*l and *transaction-leve*l. Statement-level read consistency ensures that all of the data returned by an individual query comes from a specific point in time: The point at which the query started. This guarantees that the query does not see changes to the data made by other transactions that have committed since the query began. This is the default level of read-consistency provided by Oracle.

In addition, Oracle offers the option of enforcing transaction-level read consistency. Transaction-level read consistency ensures that all queries made within the same transaction do not see changes made by queries outside of that transaction but can see changes made within the transaction itself. These are known as *serializable* transactions.

Temporary segments are used as temporary workspaces during intermediate stages of a query's execution. They are typically used for sort operations that cannot be performed in memory. The following types of queries may require a temporary segment:

- SELECT.....ORDER BY
- SELECT.....GROUP BY
- SELECT.....UNION
- SELECT.....INTERSECT
- SELECT.....MINUS
- SELECT DISTINCT.....
- CREATE INDEX....

Tablespaces group related logical entities or objects together in order to simplify physical management of the database. Tablespaces are the primary means of allocating and distributing database data at the physical disk level. Tablespaces are used to:

- Control the physical disk space allocation for the database
- Control the availability of the data by taking the tablespaces online or off-line
- Distribute database objects across different physical storage devices to improve performance
- Regulate space for individual database users

Every Oracle database contains at least one tablespace named SYSTEM. The SYSTEM tablespace contains the data dictionary tables for the database used to describe its structure.

Schema objects are the logical structures used to refer to the database's data. A few examples of schema objects would be: Tables, indexes, views, and stored procedures. Schema objects, and the relationships between them, constitute the relational design of a database.

4.2.3 Physical storage structures

An Oracle database is made up of three different types of physical database files: *Datafiles, redo logs, and control files*.

An Oracle database must have one or more datafiles in order to operate. Datafiles contain the actual database data logically represented in the form of tables or indexes. At the operating system level, datafiles can be implemented as either JFS files or raw devices. The data contained in the

datafiles is read from disk into the memory regions as described in Section 4.2.1, “Memory structures” on page 68.

An Oracle tablespace is comprised of one or more datafiles. A datafile cannot be associated with more than one tablespace, nor can it be used by more than one database. At creation time, the physical disk space associated with a datafile is pre-formatted but does not contain any user data. As data is loaded into the system, Oracle reserves space for data or indexes in the datafile in the form of extents.

Redo logs are used by Oracle to record all changes made to the database. Every Oracle database must have at least two redo logs in order to function. The redo log files are written to in a circular fashion; when the current online log fills up, Oracle begins writing to the next available online redo log. In the event of a failure, changes to the Oracle database can be reconstructed using the information contained in the redo logs. Due to their importance, Oracle provides a facility for mirroring or *multiplexing* the redo logs so that two (or more) copies of the log are available on disk.

The control file describes the physical structure of the database. It contains information, such as the database name, date, and time the database was created and the names and locations of all the database data files and redo logs. Like the redo logs, Oracle can have multiple copies of the control file to protect against logical or physical corruption.

4.2.4 Processes

A process is defined as *a thread of control* used in an operating system to execute a particular task or series of tasks. Oracle utilizes three different types of processes to accomplish these tasks:

- User or client processes
- Server processes
- Background processes

User processes are created to execute the code of a client application program. The user process is responsible for managing the communication with the Oracle server process via a *session*. A session is a specific connection of a user application program to an Oracle instance. The session lasts from the time that the user or application connects to the database until the time the user disconnects from the database. The processes connected to an instance include the Oracle SID in their process name.

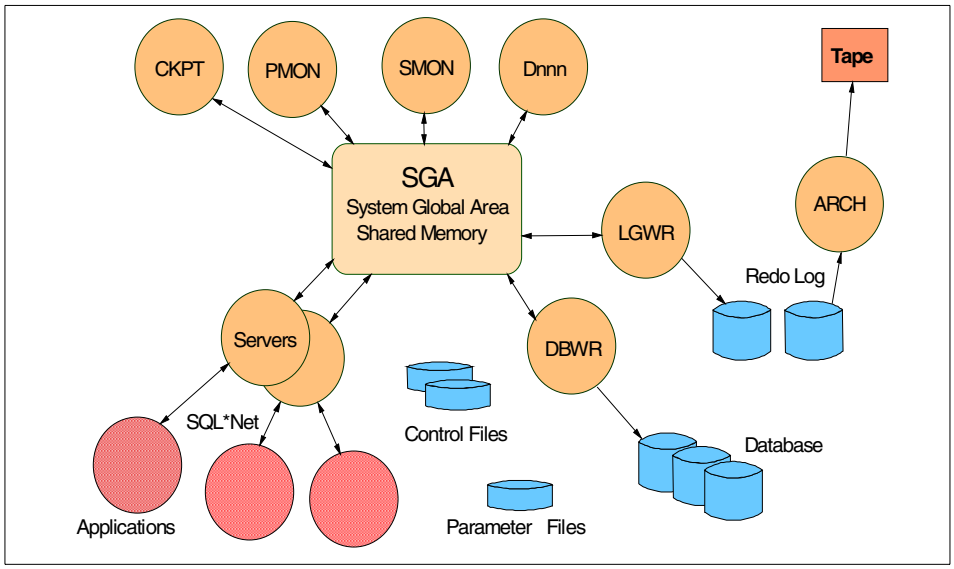


Figure 16. Oracle architecture

Server processes are created by Oracle to service requests from connected user processes. They are responsible for interfacing with the database to carry out the requests of user processes. The number of user processes per server process is dependent on the configuration of Oracle. In a *dedicated server* configuration, one server process is spawned for each connected user process. In a *multi-threaded server* configuration, user processes are distributed among a pre-defined number of server processes.

Oracle background processes are created upon database startup or initialization. Some background processes are necessary for normal operation of the system, while others are only used to perform certain database maintenance or recovery related functions. The Oracle background processes include:

- **Database Writer (DBW*n*)**
  The database writer process is responsible for writing modified or *dirty* database buffers from the database buffer cache to disk. It uses a least recently used (LRU) algorithm to ensure that the user processes always find free buffers in the database buffer cache. Dirty buffers are written to disk using a single multi-block write. Additional database writer processes can be configured to improve write performance if necessary. On AIX, enabling asynchronous I/O eliminates the need for multiple database writer processes and should yield better performance. Please reference

13.9.1, “AIX asynchronous I/O” on page 318 for additional information regarding the use of asynchronous I/O.

Note

Multiple database writer processes provide no performance benefit on uniprocessor based systems.

- **Log Writer (LGWR)**
  The log writer process is responsible for writing modified entries from the redo log buffer to the online redo log files on disk. This occurs when one of the following conditions is met:
  - Three seconds have elapsed since the last buffer write to disk
  - The redo log buffer is one-third full
  - The DBW*n* process has written modified buffers to disk
  - A transaction commits

  A *commit record* is placed in the redo log buffer when a user issues a `COMMIT` statement, at which point, the buffer is immediately written to disk. The commit record serves as a reminder to the LGWR process that the redo entries associated with this particular transaction have already been written to disk. The actual modified database data blocks are written to disk at a later time, a technique known as *fast commit*. The committed transaction is assigned a *system change number (SCN)*, which is recorded in the redo log in order to uniquely identify the changes made within the transaction.
- **Checkpoint Process (CKPT)**
  The checkpoint process is responsible for notifying the DBW*n* process that the modified database blocks in the SGA need to be written to the physical datafiles. It is also responsible for updating the headers of all Oracle datafiles and the controlfile(s) to record the occurrence of the most recent checkpoint.
- **Archiver (ARCH)**
  The archiver process is responsible for copying the online redo log files to an alternate physical storage location once they become full. The ARCH process exists only when the database is configured for `ARCHIVELOG` mode.
- **System Monitor (SMON)**
  The system monitor process is responsible for performing recovery of the Oracle instance upon startup. It is also responsible for performing various

other administrative functions, such as cleaning up temporary segments that are no longer in use and coalescing free extents.

- **Process Monitor (PMON)**
  The process monitor is responsible for cleaning up after failed user processes. This includes such tasks as removing entries from the process list, releasing locks, and freeing up used blocks in the database buffer cache associated with the failed process.
- **Recover (RECO)**
  The recover process is responsible for recovering all in-doubt transactions that were initiated in a distributed database environment. RECO contacts all other databases involved in the transaction to remove any references associated with that particular transaction from the pending transaction table. The RECO process is not present at instance startup unless the DISTRIBUTED_TRANSACTIONS parameter is set to a value greater than zero.
- **Dispatcher(D*nnn)***
  Dispatcher processes are only present in a multi-threaded server configuration. They are used to allow multiple user processes to share one (or more) server processes. A client connection request is received by a network listener process, which, in turn, passes the request to an available dispatcher process who then routes the request to an available server process. If no dispatcher processes are available, the listener process starts a new dedicated server process and connects the user process directly to it.
- **LOCK (LCK*n*)**
  The lock process is used in Oracle Parallel server configurations to ensure inter-instance locking. As of Oracle 8, up to ten lock processes can be started.

4.2.5 SQL extensions - Stored procedures

Oracle has extended the standard ANSI/ISO SQL specification by providing additional tools and technologies for accessing the database. These include both the procedural language extension PL/SQL and the support for embedded SQL in Java language programs via SQLJ.

4.2.5.1 PL/SQL

PL/SQL is Oracle's proprietary programming language extension to the SQL language. It allows the user to define such common programming language constructs as procedures, packages, and functions, to manipulate data stored in the Oracle database. PL/SQL is a *block-structured* language, meaning that the procedures and functions that make up the program are divided into

logical blocks. The PL/SQL engine is available not only in the Oracle server, but in application development tools, such as Oracle Forms and Oracle Reports as well. PL/SQL has the following features and advantages:

- SQL support.
- Object-oriented support.
- Portability. It will run unmodified on any platform that Oracle supports.
- Increased performance.
- Tight integration with Oracle.

The PL/SQL engine is also responsible for processing Oracle *stored procedures*. Stored procedures are commonly used procedures used by an application that has been stored in the database for easy access. Some characteristics of stored procedures are that they:

- Can take parameters and return values
- Can be called by many users
- Are stored in the Oracle data dictionary
- Execute on the database server

4.2.5.2 SQLJ

SQLJ is a Java language extension that allows application programmers to embed static SQL statements in their Java code in order to extract and manipulate data in the database. Static SQL statements are predefined and do not change over the course of the execution of the program. Oracle's SQLJ implementation consists of two major components:

- Oracle SQLJ translator
  The translator is a preprocessor or precompiler that is run against SQLJ source code to produce Java file output. A Java compiler is then invoked to produce class output files from the Java source code.
- Oracle SQLJ runtime
  The SQLJ runtime is invoked when a user invokes a SQLJ compliant application. It then accesses the underlying database using the Oracle Java Database Connectivity (JDBC) driver.

4.2.6 Administration tools

Oracle provides several unique tools for managing various aspects of the database. These include tools for exporting and importing data, performance and availability monitoring, and centralized database administration. These

include *Server Manager, Export* and *Import,* and *Oracle Enterprise Manager (OEM*).

4.2.6.1 Server Manager

Server Manager is an Oracle tool used to perform routine database administration tasks, such as startup and shutdown of the database, backup and recovery, and running dynamic SQL statements. There are both command-line and GUI versions of the tool available. Server Manager can be used to:

- Create a database and initialize the Oracle data dictionary
- Administer multiple databases
- Centralize database management by connecting to both local and remote databases
- Dynamically execute SQL, PL/SQL, or Server Manager commands

4.2.6.2 Export and Import utilities

Oracle provides utilities for the exporting and importing of data contained in the database. The primary tools for exporting and importing are `exp` and `imp` commands. The `exp` command is used to extract the object definitions and table data from an Oracle database and store them in an Oracle proprietary binary format on disk or tape. Likewise, the `imp` utility is used to read the object definitions and table data from the exported data file and insert them into an Oracle database.

While the `imp` utility addresses the need for importing data that was exported from an existing Oracle database, it does not provide a facility for importing user defined data. The Oracle utility *SQL*Loader* is used to load data from external data sources into an Oracle database. SQL*Loader can be used to:

- Load data from multiple input files of different file types
- Load data from disk, tape, or named pipes
- Load data directly into Oracle datafiles, thereby, bypassing the Oracle buffers and increasing the speed of data import operations
- Load fixed format, delimited format, or variable length records
- Selectively filter data based on predefined filtering rules

4.2.6.3 Oracle Enterprise Manager (OEM)

Oracle Enterprise Manager (OEM) is an integrated system management tool for managing Oracle databases. It includes a graphical console, intelligent system agents, and access to common database management tools. The

graphical console serves as a central management point for the database allowing the database administrator to:

- Administer and tune multiple databases
- Perform software distribution to both client and servers
- Schedule jobs to run on different databases at different times
- Monitor and respond to predefined database events

Chapter 5. Parallel databases

This chapter's purpose is to provide information about the parallel versions of DB2 UDB and Oracle. These are called DB2 UDB EEE and Oracle Parallel Server (OPS). Both of these products are covered in greater detail in other redbooks and are, therefore, not the main subject matter for this redbook. They have been included here to highlight the functional differences between the standard (sometimes called classic) versions of these databases and the parallel versions.

This chapter also allows you to gain an understanding of how the parallel versions work, how they benefit from the RS/6000 SP architecture, and in which workload environments they are most efficiently used.

Parallel Database Management Systems are designed to store and manage very large databases and to provide better performance than purely serial database systems. The term Very Large Databases (VLDB) is used to refer to databases that are 200 GB or larger in size. Databases for Decision Support (DSS), Data Warehouses or OLAP typically contain a large amount of data generated over a large time frame. They usually have one, very large table called a *fact table* and some small tables called *dimension tables*. OLTP databases can have very large tables too, but this is not as common.

The implementation of parallel RDBMSs depends on the hardware architecture on which the database system runs. This chapter provides information about how parallel database concepts are implemented by DB2 UDB and Oracle on the RS/6000 SP. Some reflections about the advantages and disadvantages of parallel database systems are made.

5.1 Parallel concepts in database environments

Large and parallel databases benefit from certain system architectures, such as shared memory, shared disk, or shared nothing architectures. The implementation of parallel databases also depends on the hardware architecture on which the database system runs. This section is meant as a short introduction into these system architectures.

5.1.1 Shared memory

In a shared memory environment, the system consists of two or more processors.These multiple processors access the same memory and also the disks of the system concurrently. This is called a Symmetric Multi Processor (SMP) environment. The database system uses the availability of multiple

processors to split the workload of a query onto these CPUs in order to improve the query's response time. This is shown in Figure 17 and is typical of the RS/6000 R Series, H Series, and S Series machines.

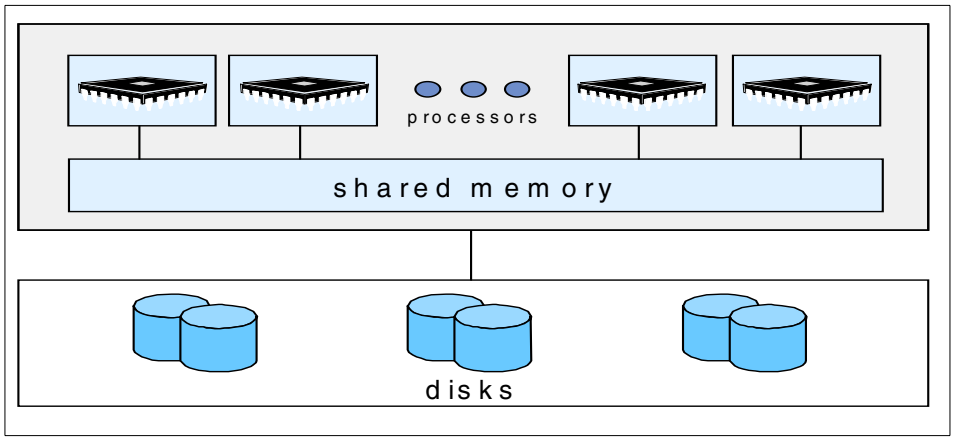


Figure 17. Shared memory

5.1.2 Shared disks

In a shared disk environment, every CPU has its own dedicated memory, but all processors share the same disks within the system. An RDBMS on this system consists of one database system that stores data and indexes across all disks. Every process running on any CPU has access to all data and indexes that are placed on all disks of the system. In order to improve performance, it is easy to add additional CPUs and memory. Therefore, systems with shared disk architecture have good scalability. This is shown in Figure 18 and is used in RS/6000 HACMP clusters for fast recovery.

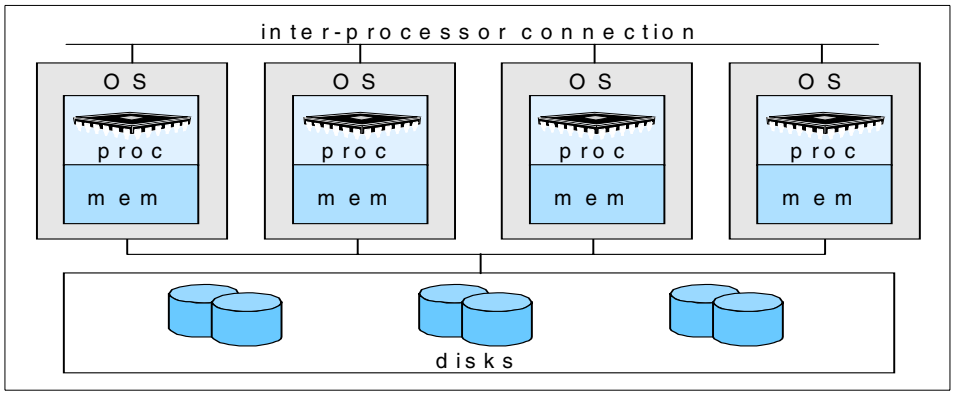


Figure 18. Shared disk

5.1.3 Shared nothing

In a shared nothing environment, every CPU has its own memory and its own set of disks and, therefore, does not have to compete for resources with other processors. These *loosely coupled* systems are linked by a high speed interconnection (see Figure 19). As nothing is shared, the system can scale to a high number of nodes. This environment is referred to as Massively Parallel Processors (MPP). It can be implemented by two or more separate RS/6000 systems connected via fast communication adapters (Token Ring, Ethernet).

An implementation of this concept is the RS/6000 SP system. This system has several nodes installed in one or more frames. A node is an independent RS/6000 model with CPU, memory, internal disks, and communication adapters. The nodes are connected via the SP Switch for inter-communication. Within the RS/6000 platform, parallel databases means RS/6000 SP.

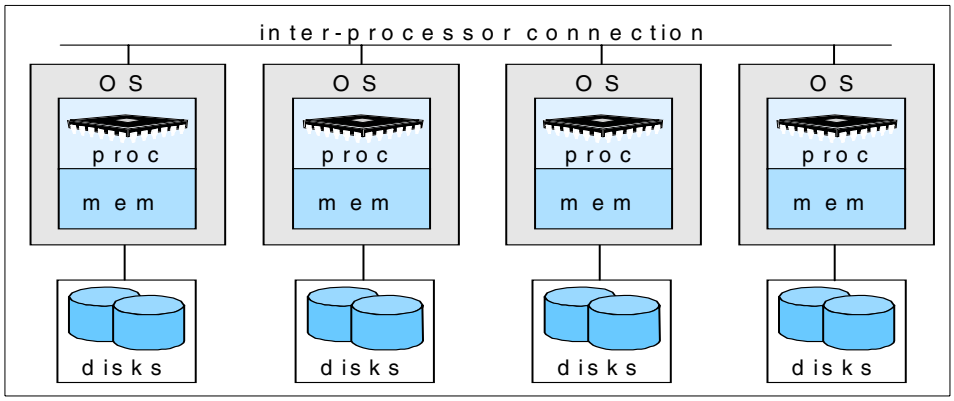


Figure 19. Shared nothing

5.2 DB2 UDB Enterprise - Extended Edition (EEE)

IBM implementation of parallel database concepts on AIX is DB2 Universal Database Enterprise - Extended Edition (DB2 UDB EEE). This product is based on the same code as DB2 UDB Enterprise Edition and extended with the fileset *DB2 UDB Parallel Extension* that enables the distribution of data and indexes over multiple database partitions. For this reason, it is also called a *partitioned database system* or *clustered system*.

5.2.1 Concepts and functionality

What makes DB2 UDB EEE different from DB2 UDB EE is the capability of splitting the data into *database partitions*. A database partition is a logical concept that must be mapped to the available hardware by the database administrator. On an RS/6000 SP, you may decide to assign one partition to every single node of your system, but it is also possible to declare several partitions per node. On large SMP machines, you logically assign a number of processors, memory, and a couple of disks per partition. The decision of how many partitions your system should have must be a balance between the overhead a partition adds to your configuration (in terms of CPU, memory, and administrative resources) and the higher level of parallelism you gain for some operations, such as loads, backups, and all those workloads that are often referred to as *batch jobs*. In previous versions of DB2 UDB, database partitions were called *logical nodes*, which gives an idea of the internal processing independence that one database partition owns.

On each database partition, one database manager is responsible for a portion of the databases data. Each database server has its own set of data.

The fact that data is partitioned across database partition servers is transparent to users and applications. Each database system has one database partition on which the `create database` command for the database was issued. This database partition contains the database system catalogs for the entire database and is called the *catalog node.* The user interaction with the database is handled from the node the user is connected to. This database partition is known as the *coordinator node.* Each database partition can act as a coordinator node to handle the distribution of system resources.

Data and indexes are stored in tablespaces. DB2 UDB EEE uses nodegroups to define to which nodes every table space is distributed. A nodegroup is a group of database partitions.

DB2 UDB EEE executes everything in parallel. All database functions, such as `SELECT, INSERT, UPDATE`, and `DELETE` are performed in parallel on all database partitions. Both database activities, such as data scan, index scan, joins, sorts, index creation, or table reorganization and DB2 UDB utilities, such as data load, backup, and restore are executed simultaneously on all partitions. Particularly the loading of very large amounts of data can be performed much faster in this parallel database environment than on a serial database system.

DB2 UDB EEE provides excellent scalability. If the number of concurrent connected users grows over time, or the amount of data reaches the system resource capacity, a parallel database system can be extended by adding additional nodes to the system and defining new database partitions. DB2 UDB EEE provides the capability to redistribute the data onto the newly added node. This means that after adding a database partition, the database manager starts to move a part of the data to the new node in order to get an equally balanced system. This process is referred to as *data redistribution*.

Communication across all database nodes is realized by the *Fast Communication Manager* (FCM). Each database partition has one FCM daemon to provide communication support in order to handle DB2 UDB agent requests and to manage message buffers. In an SP environment, the FCM daemons interact over the SP Switch and TCP/IP sockets. On an SMP system, this communication happens in shared memory.

DB2 UDB EEE uses a hashing strategy to partition the data. If you want a partitioned table, you have to decide on which nodes (database partitions) the data will be distributed. These nodes will form a nodegroup that can be reused for other tables you want to partition as well. Internally, DB2 UDB will create for you a *partitioning map,* which plays an essential role for the hashing process. Then, you have to declare a tablespace and assign it to the nodegroup previously

created. It will contain the definition of the physical layout, that is, you specify directories, files, and raw devices where the data will be physically stored. Again, this tablespace can be reused for other tables and indexes as well. Finally, you declare your table in this tablespace, specifying a *partitioning key* (the second essential player in the hashing process) and start to insert or load data. DB2 UDB will decide to which node each record must be sent.
The partitioning key consists of one or more columns of the table. Do not forget that all columns of the partitioning key must be included in the primary key or unique indexes you might want to create on this table. Briefly, the partitioning process starts when a new row must be inserted: DB2 UDB applies its internal hashing function, taking the value(s) of the column(s) defined as partitioning key argument(s), and produces a *hash-value* for this row. Then, DB2 UDB scans the partitioning map of the nodegroup searching for the hash-value. The partitioning map contains the information on which node is assigned to which hash-value. Finally, the row is sent to the node according to the definition of the partitioning map and is stored in the datafiles specified in the tablespace.

The determination of the best partitioning key is important for the most effective distribution of rows and is, therefore, essential for optimal system performance. See also *The DB2 Cluster Certification Guid*e, ISBN 0-1308-1500-X, for more information.

5.2.2 Optimizer

DB2 UDB EEE uses a cost-based optimizer. It compares different data access methods and selects the most efficient one. The optimizer uses information about how base tables and the intermediate tables that result from queries are partitioned across the system and determines the best execution strategy. As a result, the optimizer determines a cost optimized access plan for that query that can be visualized through performance tools, such as Explain. When generating the access plans, the optimizer considers different parallel methods for joining tables including *co-located, directed,* and *broadcast* joins. The optimizer features extensions, such as SQL query rewrite, SQL extensions, Star Joins, Dynamic Bit-Mapped Indexing ANDing (DBIA), and OLAP extensions to find the optimal strategy for joining very large tables with one or more small table.

5.2.3 Inter-partition and intra-partition parallelism

All database operations, such as index scans and table scans, aggregation, set operations, joins, inserts, deletes, and updates can gain significant performance improvements provided by the *intra-query parallelism* and/or *inter-query parallelism*. The DB2 UDB cost-based optimizer decides whether a user statement runs in parallel or not, which stages of that user statement

are parallelized, and how these stages are parallelized. The decision is based on:

- Available hardware (such as the number and speed of processors, the number of nodes, and the number and speed of disks)
- Configuration parameters
- Current workload (how many queries run on the system and how many resources are consumed)
- Type of operation
- Physical layout (nodegroup configuration, tablespace containers)
- Available information about the objects referred to in the statement (table and index statistics)

Inter-partition parallelism means that the function is executed in parallel by each database partition. An example would be when a user or application issues an SQL statement, such as a SELECT statement, to fetch data with certain conditions from many tables spread over multiple nodes. In this case, the coordinator node sends this request to all database partitions. The database manager on each node selects the data from tables stored on the disks, sorts the data, and sends all rows that meet the selected conditions back to the coordinator node. On this node, all rows are finally merged and returned to the user or application. In this example, the function (query) is shipped to all nodes, and only the data that satisfies this request is sent back across the network. This concept reduces the network traffic and is known as *function shipping.*

Intra-partition parallelism allows different operators in the same query to be executed in parallel by the same database partition node. If, for instance, an application performs a query including a `SCAN`, a `JOIN`, and a `SORT`, the database manager can execute this request in parallel depending on the setting of dedicated DB2 UDB configuration parameters.

5.2.4 Hardware implementation

As mentioned above, DB2 UDB EEE is designed for two different types of hardware configurations:

- MPP systems with shared nothing architecture
- SMP systems with shared memory architecture

5.2.4.1 DB2 UDB EEE on RS/6000 SP

An RS/6000 SP is the most commonly used system for DB2 UDB EEE. Each node is an independent RS/6000 model with CPU, memory, internal disks,

and several adapters for communication, external disks, and devices. One key part of the SP is the switch, which is responsible for high performance data transfer. The communication protocol used by the nodes to communicate through the switch is TCP/IP with the advantage of easy implementation and configuration associated with this protocol. The administration of this system is done from a single point of control using a dedicated machine known as Control Workstation (CWS). This is an RS/6000 model with enough disk space to hold all management tools and all LPP sources for all nodes. It also contains a graphic adapter, monitor and keyboard, a CD ROM for software installation, and backup devices, such as tape drives.

Additional software must be installed on all nodes and the CWS in order to provide inter-communication support and management. This software is the AIX Parallel System Support Programs (PSSP), which is also responsible for authentication, security, and administration.

This environment is best suited to make use of all the features that DB2 UDB EEE provides for optimal performance.

If an SP system contains SMP nodes, it is possible to combine the advantages of MPP systems with those of SMP systems. In this environment, two or more database partitions reside on one SMP node, and each node is part of a clustered databases system on the SP.

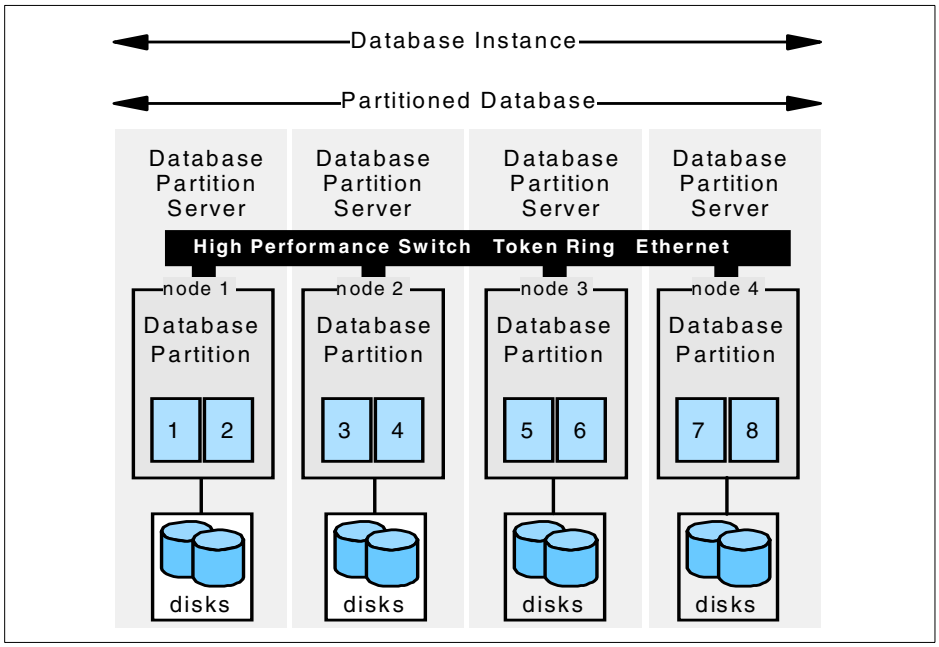


Figure 20. DB2 UDB EEE structure

5.2.4.2 DB2 UDB EEE on an SMP system

DB2 UDB EEE installed on an SMP system corresponds to the concept of a shared memory implementation. Optimal performance will be achieved by using a system with eight or more CPUs, on which one database partition uses one, two, or four processors. The amount of memory should be as large as possible to provide enough space for buffer pool and buffers for all database manager processes. In order to prevent the disk subsystem from becoming a bottleneck for I/O throughput, it is recommended to use more than one SCSI adapter or more than one SSA subsystem to provide a larger bandwidth.

This environment might be a good solution for DSS databases, BI applications, and Data Marts where many complex queries run in a loop against a larger table, and the most used computer resource is the CPU.

5.3 Oracle Parallel server

Oracle has implemented its RDBMS product on the IBM RS/6000 SP so that it can run on multiple nodes of an SP in parallel. This implementation uses the shared disk model as discussed earlier in this chapter and is unique in

this as the other parallel databases use shared nothing. The parallel version of Oracle is like classic Oracle with a few additions.

- It makes use of the Oracle Parallel Query (OPQ) features that are also available on classic Oracle, but here, not only does the query get split out onto the CPUs of a single system but also split out across the nodes of the SP. This feature is used a lot in DSS workloads where all the CPUs in all of the different nodes can participate in working on the query for maximum parallelization and reduced response times.
- The addition of Oracle Parallel Server (OPS), which allows different instances of Oracle to run on different nodes of the SP having shared access to a single database. OPS uses two extra subsystems to achieve this: The Virtual Shared Disk (VSD) and the Distributed Lock Manager (DLM). These two components are described below.
- Many Oracle tools have been enhanced to allow extra parallelization, for example, parallel load and parallel indexes.
- To support very large databases, the new partitioned tables features is very important. It reduces DBA time for load, index, and delete operations of very large tables and can also reduce query time.

Many of these extra features are also available in the classic version of Oracle where they are useful, but they become very important for OPS and with extremely large databases.

5.3.1 Parallel Oracle architecture

Figure 21 shows the various components of classic Oracle. There are two major components:

1. The Instance - The processes connected and co-operating via the SGA. The *front end* processes (also referred to as client processes and server or shadow processes) are connected to users and execute the SQL statements. The *back end* processes (also referred to as background processes) are internal to Oracle for the redo log, database updating and recovery. They come and go with an Oracle Instance.
2. The Database - All the disks and files that make up the database.

See 4.2, “Oracle database architecture” on page 68 for more details on Oracle structure.

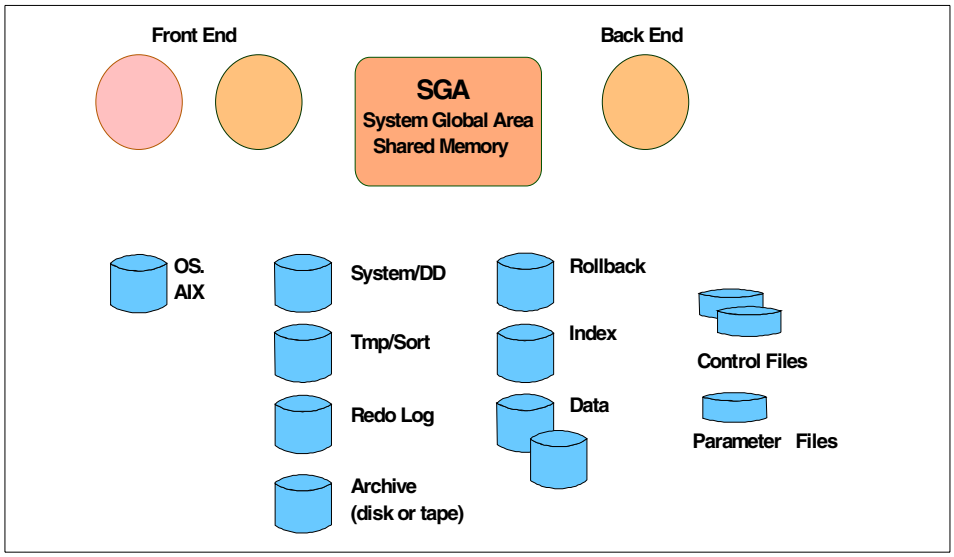


Figure 21. An Oracle instance

In Oracle Parallel Server, there are multiple copies of the instance and a single database. Figure 22 shows the overview level of Oracle with multiple instances and one database. The database is, of course, made up of lots of files and/or devices that are ultimately on a disk in the system. This general architecture of the OPS is then mapped to the SP architecture where each SP node is an RS/6000 computer on its own. Each node has:

1. One CPU or multiple CPUs (in an SMP node)
2. Memory
3. Adapters
4. Disks

On the SP, there is also the High Speed Switch network that allows fast communication (low latency) and high bandwidth (large data transfers) between the nodes of the SP.

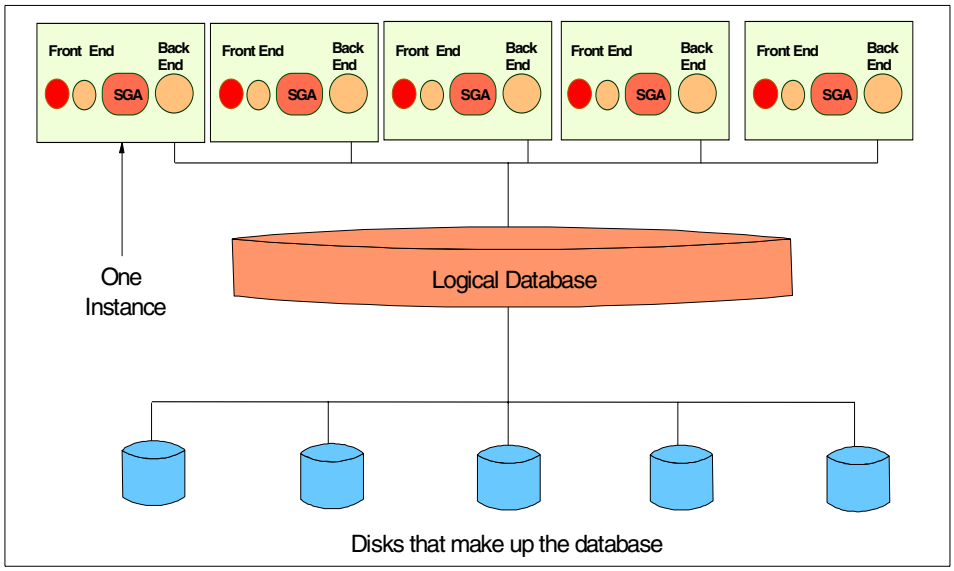


Figure 22. General Oracle Parallel Server architecture

With OPS, one instance of Oracle is run on each node of the SP, but there are two problems with this architecture:

1. An individual disk is actually attached to one of the nodes. This makes it impossible for an instance on one node to read data from disks that are attached to another node.
2. All of these instances of Oracle may have local copies of data blocks, and there is a risk of two of them updating the same data and, thus, corrupting the database.

These two problems are addressed by two software systems:

1. The *Virtual Shared Disk* (VSD) makes it possible for OPS to read the data from a remotely attached disk. The details are in 5.3.2, “Virtual Shared Disk (VSD)” on page 94.
2. The *Distributed Lock Manager* (DLM) makes sure data corruption between instances does not happen.The details are in 5.3.3, “Distributed Lock Manager (DLM)” on page 95.

Figure 23 shows the architecture of OPS when implemented on the SP. The database is spread across disks that are attached to all the nodes of the SP. This spreads the I/O requirements evenly across disks, adapters, and all of the nodes in order to reduce I/O bottlenecks.

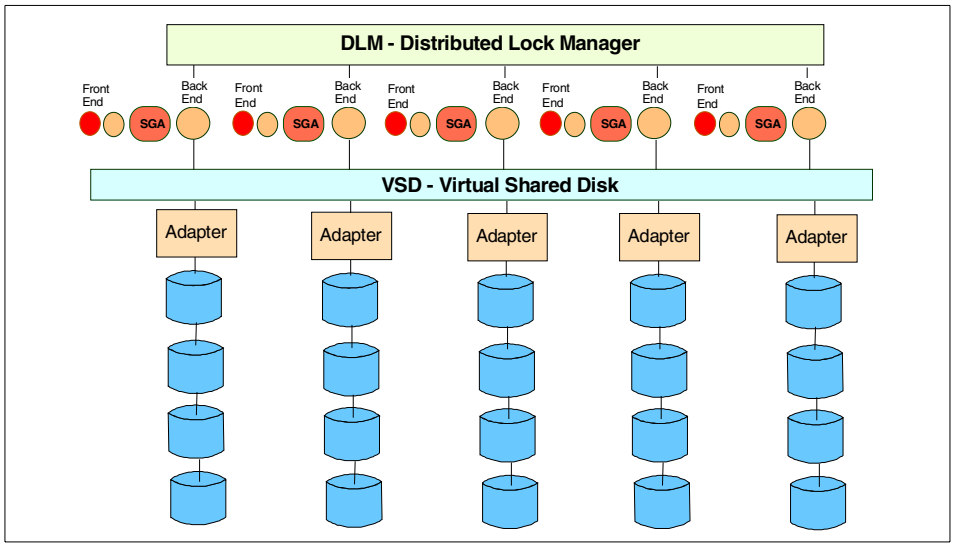


Figure 23. Parallel Oracle on SP

When an application connects to OPS, it actually connects to one of the instances. As every instance has access to the entire database, simple (OLTP type) queries are performed directly by that instance. If the query is large, Oracle is able to split the query into sub-queries. For example, a full table scan can be broken down into a number of sub-queries that each work on a range of rows, and other sub-queries can merge the results together and sort the results. On an SMP node, these sub-queries can be run on different CPUs. For very large queries, the sub-queries can also be sent to other nodes of the SP, and the results are sent back to the original node. So, in Parallel Oracle, there are two types of parallelization:

- Degree - The number of processes (and, therefore, CPUs) on an instance
- Instance - The number of instances to use

The decision on the amount of parallelization is made by the optimizer and is based on the parameters set on one of the following:

- Table level parallelization settings
- Hints in the SQL statement

In general all the nodes in an OPS system on the SP are kept identical to reduce system administration and DBA workloads. Due to the extra dimension of multiple nodes and the large size of these databases, OPS is considered complex when compared to a single SMP machine running on

smaller databases with classic Oracle. However, the power and scalability of the SP does mean that much larger databases can be implemented, and this is a requirement, particularly for DSS workloads.

5.3.2 Virtual Shared Disk (VSD)

This is an IBM supplied product that allows any of the nodes in the SP to read and write data from disks attached to other nodes. Figure 23 shows that the VSD layer is between the Oracle Instance and the device drivers and disk adapters. OPS does not have direct access to the disks but opens VSD devices instead. The VSD layer then checks if the real disk is attached to the local node, and if not, it works out which other node has it attached:

- In the case of a local disk, the read/write request is passed by the VSD directly to the regular logical volume device driver on the node.
- In the case of a remote disk, the request is sent by the VSD device driver (via the SP Switch) to the VSD device driver on the node to which the disks is attached. The receiving VSD device driver then passes the request to the device driver of the second node, and the data is read or written on behalf of the initial node. The data is also transferred over the SP Switch.

The difference in time between the local and remote disk I/O is small.

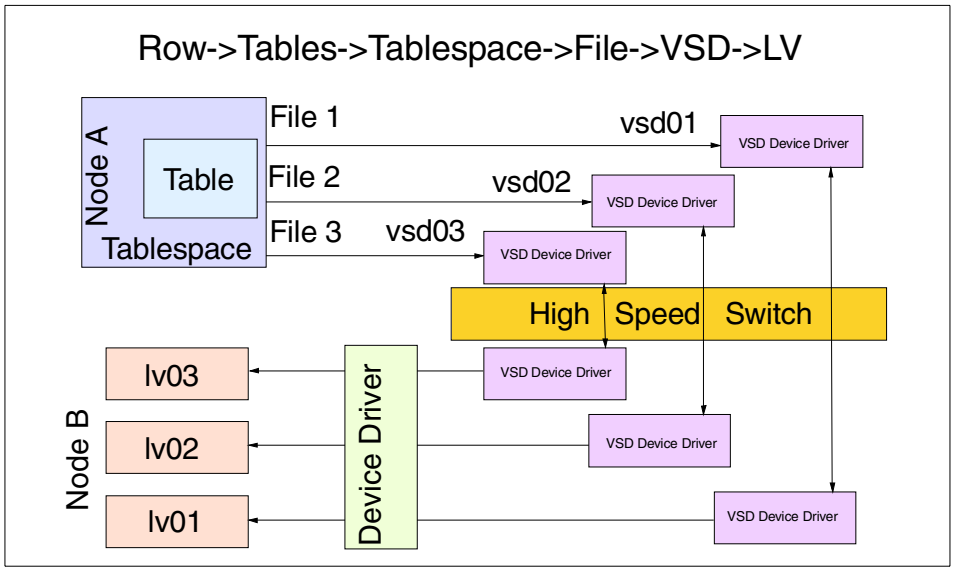


Figure 24. Oracle tables working with VSD operations

Figure 24 shows the VSD structure in more detail. Node A thinks it has three files that make up a tablespace, but the files are actually VSDs. When Oracle reads from or writes to these files, the requests are passed over the High Speed Switch to the right node.

In OPS, all Oracle data files are raw devices; there are no JFS based data files except for the init.ora parameter files. Reading from, and writing to, a table means I/O operations are performed to one or more of the files that make up the tablespace in which the table resides. As far as Oracle knows, these files behave like real devices but are actually VSD devices. If necessary, the disk I/O requests are redirected to the node with the logical volumes that contain the actual data where the regular logical volume manager device driver actually does the disk I/O.

5.3.3 Distributed Lock Manager (DLM)

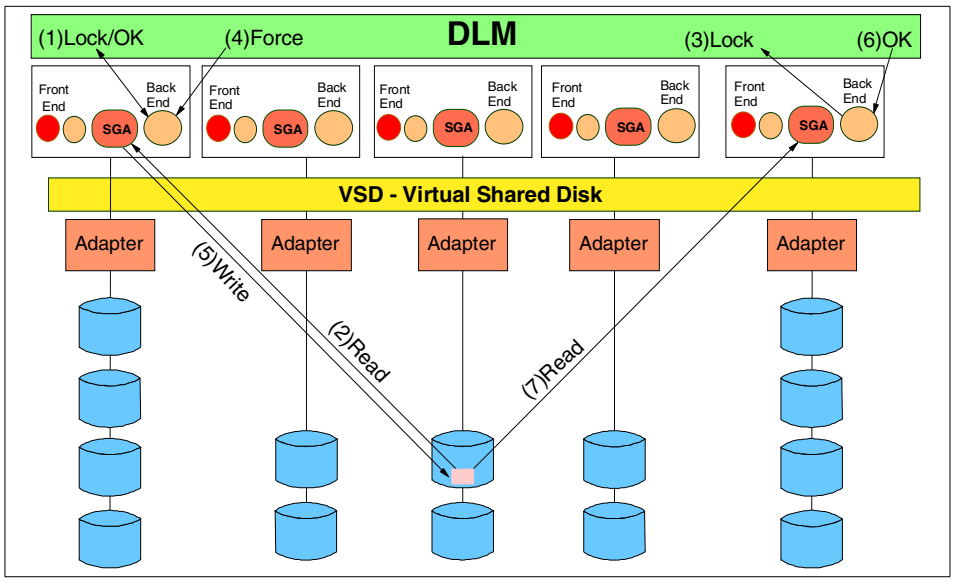


Figure 25. Distributed Lock Manager operation

The following is a brief description on how the DLM works by taking a simple example where the instances on the far right and far left want to update the same block in the database. The order is as follows:

1. The right instance wants the block; so, it requests a lock from the DLM for that block, and the DLM grants the lock because no other instance has the lock.

2. The right instance reads in the block into its local SGA buffer cache.
3. The left instance wants the block; so, it requests a lock from the DLM for that block, but the DLM finds that the block is locked.
4. The DLM requests the right instance to release the lock.
5. When the right instance has finished the transaction, it writes the block to the disk via the VSD and informs the DLM that it no longer needs the lock and releases it.
6. The DLM grants the lock to the left instance.
7. The left instance reads in the block.

Note that this example is for instances needing to update a block. In the case of read access, multiple instances can have a read-only copy of a block, but if an instance wants to update the block later, they all have to release the read copy.

Improvements have been made in Oracle 8i (Version 8.1.5 and later), which improve OPS performance by eliminating the need for two disk operations when one instance needs a read-only copy of a data block that is modified in another instances cache.

In this case, a ping (as described above) is replaced with a cache to cache transfer of the data block over the switch or network. This is referred to as *Cache Fusion*. The cache to cache transfer over the network, along with the required DLM communication, is much faster than having one instance write the block out to disk and the other instance read it from disk.

5.4 Advantages and disadvantages of parallel databases

The overriding advantage of a parallel database is that it can be used with database sizes far bigger than the largest database that can be connected to an SMP system. Some recent RS/6000 SP systems are in the 10 terabytes to 80 terabytes range of disk space.

There are three workloads to cover:

1. For decision support queries that are very complex and involve very large volumes of data, the parallel databases allow the whole system to split down the work into small components so that the parallel machine can work on it. This gives good performance gains in reduced response times and increased throughput.
2. For medium complex queries (such as data mart, OLAP, or batch loads), where there are a number of users (for example, five to 50), the various

levels of parallelization means that the parallel database can spread the demands across a portion of the machine to give concurrent access and reduced query times. If there are high numbers of users (for example, higher than the number of CPUs in the system), then the queries are typically not made parallel. For this to work, the users of the system have to be requesting small to medium size queries.

3. For small queries (such as OLTP), there are issues for the parallel database. These evolve around three critical areas that are harder for parallel databases than for classic databases:
   - Function shipping - For DB2 UDB EEE, if the query only involves very low numbers of rows, the request is sent to the nodes with the data. The results are sent back to the original node for final manipulation to return them to the application. This requires inter-node communication and CPU resources. The classic DB2 UDB does not have this overhead and, therefore, will appear faster.
   - I/O shipping and locking - For Oracle OPS with more than three nodes, the majority of the I/O is done remotely (via VSD). This takes extra time but also involves locking the data via the DLM, which takes extra time too. In the worst case, however, every transaction requests the same block (or small set of blocks). This involves the DLM forcing the release of the lock and the block being repeatedly written to disk in order to be read into another node (called block pinging). Applications requiring this will run much slower on a parallel database.

   Therefore, for OLTP type workloads, there are problems to consider. These can be tackled with:
   - Careful design of the database and partitioning of the data.
   - Careful design of the application to reduce inter-node workload.
   - Use of a transaction processing monitor, such as: CICS, Encina, Tuxedo, or Top End to make sure the transactions go to the right node and data.

Parallel databases and large databases go hand-in-hand. This means the problems of parallel databases are mainly due to the size of the database. Large and parallel means:

- Large systems cost more, are complex to justify in the initial stages, and take longer to implement and, thus, have management focus. Therefore, they need careful and experienced project management. Many large system problems stem from lack of project management rather than technical problems.

- Large systems take higher people resources. A single parallel 5 terabyte system might take less man-power than managing 50 * 100 megabyte systems but will take more than a single 100 megabyte system.
- Parallel RDBMS and the RS/6000 SP are more complex and require that database and system administrators have additional education and experience.

Given a choice between a large SMP system or a small parallel system, we usually recommend an SMP system unless the database size will outgrow the capabilities of the largest SMP. The size of IBM SMP systems, like the RS/6000 S80, are growing every year due to the developments in processor design and SMP technology. There is some cut off between the largest SMP available today and a set of SP nodes. It can be very hard to determine where that is, but remember the extra complexity of a parallel database added to the path length for a transaction. This means more CPU instructions are executed on a parallel database. A single node parallel database will run slower than a single node classic database with the same CPU power. This negative impact is removed as soon as the parallel database has more nodes. For these reasons, we do not recommend less than four nodes in a parallel database environment.

Part 2. System design and sizing for optimal performance

Chapter 6. Sizing a database system

This chapter provides an overview of concepts and rules of thumb for sizing a database system based on IBM RS/6000.

Sizing is the estimation of the type and size of the components in a balanced system that supports a defined application workload while meeting a set of performance requirements.

System components can be, for example, CPU, memory, disks, and network connections. Application workload is determined by the type of application, the amount of data, the number of users, and the type and number of transactions and/or batch jobs the users initiate on the system. Performance requirements are usually stated in terms of expected throughput and response time.

The objective of sizing a system for RDBMS should be a balanced system. A balanced system is a system that has a sensible set of CPU, memory, disks, and network connections. There should be no bottleneck, which would make the other components ineffective. All parts of a balanced system are used appropriately during normal business operation workloads.

Accurate sizing requires detail quantitative requirements, which are rarely available, therefore, a practical approach is needed.

The final aspect is that the recommended system configuration is reflected in the costs of the machine. Successful system sizing requires choosing the right components from the alternatives to maximize the price performance and to meet the growth requirements.

Note

Sizing always assumes the system is installed properly and well tuned. A poorly tuned system will not reach the performance requirements.

6.1 Sizing constraints

Sizing a system is a difficult undertaking. The results of the sizing is the system configuration in terms of CPU, memory, adapters, and disks to perform the required computing tasks. In addition, there are other factors that also effect the recommendation. These include:

- Growth - Most systems are expected to grow by a factor from 30 percent to 100 percent a year. This is often built into the sizing, and the recommended system must demonstrate the ability to be upgraded.
- Availability - All systems are regarded as important. But some are vital for the commercial success of the company and can justify the expense of disk protection and products to ensure minimum down time, such as High Availability Clustered Multi-Processors (HACMP) from IBM.
- Cost limits - Often a sizing is made to a budget. In this case, the sizing determines the best configuration for the requirement at the price or that it simply cannot be done (and requires further negotiating).

We recommend you to use the following:

- For growth, it is best to simplify the work by only considering one size at a time. Once all the configuration details are decided, then the alternative sizes are considered - one by one. It does not matter if you start with the initial or final configuration. It can be the initial system followed by how to upgrade to the final size. Alternatively, you can size the full configuration and then decide what parts can be removed to make the initial system. In either case, develop a growth plan for minimum disruption of the system during the upgrade. SMP machines make this relatively simple. Also, sometimes the initial machine can be assigned a different task as part of the upgrade. For example, it may become the test or development machine once the larger production machine is available.
- For availability, first size the system without disk protection. Once a good configuration can be recommended, the disk protection should be considered. For disk protection, the common solutions are mirrors (doubling the cost) and RAID 5. Then the additions required for HACMP should be priced. Most customers prefer to understand the extra costs and benefits as a separate issue.

Note

Size the basic system first and then add extra complexities of growth and availability to the configuration.

When planing a database system on a new RS/6000 or migrating an existing database system to a new one, several factors need to be considered. These are mainly:

- The type of workload and the CPU power required for the various transactions
- The expected size of the database data and the largest table in it

- The other database objects, such as indexes and temporary space
- The type of data you will be using
- The maximum number of concurrently connected users
- The maximum number of concurrent jobs
- The number of transactions within a certain period of time

These factors are used to determine which hardware system is needed, particularly if a single CPU system or an SMP system is required.

Unfortunately, most database sizing tasks do not have sufficient information to allow an accurate sizing. The computer saying of *Garbage In = Garbage Out* applies especially to sizing when not enough accurate details are available. However, even with very limited information, sizing does have to take place for sales and budgeting, but the unavoidable result is high error margins.

Garbage In = Garbage Out

With limited input facts, sizing becomes estimation and guesswork.

6.2 Sizing techniques

Sizing a system is a difficult task. It is not difficult though when given precise details of the requirements to precisely workout every detail of the system and the exact configuration. The problem is the level of details available from which to size. You should have the following details:

- Disk size
- Disk I/O
- The numbers of users
- The various transactions and batch processes in terms of CPU requirements
- The transaction rates
- The memory for the system (database cache and per user)
- Availability requirement to decide the needs for disk protection and HACMP
- The network latency and bandwidth
- The backup requirements

Given all of the above, a spreadsheet could quickly determine an ideal system.

But, in practice, people are asked to size based on one, two, or three of the facts above, which means a lot of guesswork and assumptions are required.

Given the lack of information, the error margin of the sizing is going to be large, but customers need some indication of the size and type of machine they should consider.

6.2.1 Sizing from the data size

Data size is often the only clear fact from which to size.

First, you have to decide if this size is the data or disk size. This has to be checked and confirmed, or the system could be wrong by a factor of three. If no other details are available, then the 1:3 ration of raw data size to disk size is used. For example, 50 GB of raw data will require 150 GB of disk space. The minimum number of disks for a database is six disks, and you should always use the smallest disks available, as the more disks you have, the higher the I/O rate.

Then, you can use the balanced systems rules of thumb to work out the CPU and memory sizes. See 6.7, “Balancing a system via the component costs” on page 119 for more information.

Be aware that the below are very rough estimation rules of thumb and should be considered to have a plus or minus 50 percent error margin.

- Memory can be very roughly sized as 5 percent of the raw data size plus 64 MB for AIX and RDBMS processes.
- CPU can very roughly be sized via the formula 2 GB of raw data per Relative OLTP. For example, the F50 4 way 332 MHz has a Relative OLTP value of 32; so, using the formula, this system should roughly be able to support 64 GB of raw data.

6.2.2 Sizing from transaction rates

Some sizing is done from the customer knowing the transaction rates that are required on the system. This is often based on previous sales and marketing estimates.

There is a serious problem when taking a years worth of transactions and working out a transactions per minute rate from this.

For example, it might be known that last year 1,000,000 items were sold and each required two transactions on the system (one for ordering and one for shipment). This gives us 2,000,000 transactions. If we allow five days a week, 52 weeks a year, and eight hours a day, we can workout that 2,000,000 / 5 / 52 / 8/ 60 = 16 transactions per minute. But people do not order or work like this. First, there are seasonal buying patterns. Second, people do not work eight hours a day at a constant rate. So, we might guess 30 percent of sales are in one month, and 60 percent of the orders are taken between 10 am and 12 mid-day. If you check the math, you get 150 transactions per minute. You can see 150 is a large difference from 16 transaction per minute. If the workload is generated from Web users, the peaks in demand can be extreme and totally unpredictable.

Once we have a transaction rate we can then compare this to some standard benchmark numbers. But be careful when comparing to standard benchmarks, such as TPC-C, TPC-H, or TPC-R. These benchmarks are run on very highly tuned system. In TPC-C, the application is very small, and the transactions are very small. If the application you are going to use, and the transactions are not exactly like the benchmark you are comparing with, you will introduce are very large margin of error.

Internally to IBM, and for IBM Business Partners, there are tools that contain typical transactions from benchmarks and are based on real production workloads. These tools are called FastSize and the RDBMS Sizer and are available from `http://w3.aixncc.uk.ibm.com` or the ParterInfo system for Business Partners. These tools have the CPU, disk read/write, and memory requirements for various transactions. This is multiplied with the number of users and the transactions rates to estimate the CPU, disk, and memory. This is then matched against the machines in the RS/6000 range to find suitable machines and configurations.

Because transaction complexity can differ by three orders of magnitude, we cannot offer any rules of thumb. If you can find examples of the transactions running on a system and know the transaction rates, then you can workout the CPU power required per transaction and work from there.

6.2.3 Sizing from user numbers

If the number of users on the system is known, then the first question that needs clarification is what kind of users they are:

- Users just known by the system as defined users
- Users logged on to the system but might not be using it at the moment
- Users actually busy using the system in the peak period

Once this is worked out, it is best to classify the users into different types, such as those only looking up data, those inputting information, those doing complex transactions, and those starting large report generation type tasks. Each of these users is likely to have different response time requirements.

We suggest that you do not think in terms of *all transaction must take less than three seconds* because there will be some large transactions that are never going to be that short. A wildcard search of a large number of records takes time and so do reports that need to summarize a lot of data. A better way to specify the required response time would be: *90 percent of the transactions will take less than two seconds,* or something similar.

Given the above information, the transaction per second or minute can be estimated, and then you can try sizing based on the transaction rate (see Section 6.2.2). The tools mentioned there can help estimate system sizes from the number of users too, but assumptions on the transaction types and rates introduce large margins of error.

6.3 Sizing for a particular application

There are many applications available on AIX and some very popular ones in the Enterprise Resource Planning (ERP) arena, for example, Oracle Financials, SAP, BAAN, and Powersoft. If you are sizing a system for an application, the first place to call is the application vendor. They, after all, understand:

- The application and the workload it generates.
- The database structure and size.
- Typical user and transaction rates.
- How the system operates in installed production systems.

Many software application vendors perform benchmarks and capacity planning tests with IBM to establish the best method of sizing and which parameters are best to estimate the machine requirements.

Many times IBM or IBM Business Partners can add experience and product knowledge once the vendor has provided the initial sizing estimates.

6.4 CPU goals and sizing

It is imperative to first determine the CPU requirements when selecting an RS/6000 model. The recommended configuration of the system should be

less than full capacity to allow for future upgrades. This is valid for all components of the system (CPU, memory, disks, etc.).

The RS/6000 family provides a wide range of systems starting from single CPU systems up to Symmetric Multiprocessor (SMP) systems with 24 processors. The selection of the system should correlate with the anticipated workload.

6.4.1 Uniprocessor (UP) Systems

A Uniprocessor System is sufficient to serve the needs of a department where the data is managed by a single database system. Memory or disks can be added, if necessary, but the number of CPUs cannot be increased. The amount of data that can be handled by a single processor is limited. As workload increases, a single CPU may become insufficient to process user requests regardless of other additional components, such as memory or disks that may be added.

6.4.2 Symmetric Multiprocessor (SMP) Systems

Symmetric Multiprocessor Systems are made up of multiple, isometric processors within the same machine. Resources, such as disk space and memory, are shared. SMP systems generally allow more disks and memory to be added than UP systems and are as easy to manage. With multiple processors, different database operations can be completed significantly faster than with a single processor.

The problem with SMP machines is that, for efficient use, each CPU needs to have something to do. This depends on the workload involved.

- Online Transaction Processing (OLTP) workload involves many users and naturally uses either a lot of processes or a lot of process threads. AIX is fully multi-threaded, thus OLTP workloads can make use of SMP machines to their full extent.
- Batch workloads usually have limited parallelisation. If the batch task is implemented as a single process, this can be a significant performance bottleneck, as only one CPU would be used. Batch implementers should endeavor to make multiple batch tasks able to run into multiple streams by splitting the task in to many parts for concurrent running.
- Decision Support Systems run a limited number of very large queries at the same time. If the number of queries is larger than the number of CPUs, then it will simply make good use of the SMP machine. If the number of queries is less than the number of CPUs, the database's parallel query

option must be used to split the query into many parts so that it can be distributed among the CPUs.

- Many DBA tasks, for example, index creation, creating summary tables and data loads, need to be parallelized. Most RDBMSs support parallel DBA tasks, and both DB2 UDB and Oracle support them.

6.4.3 CPU utilization

The CPU utilization goal should be about 70 to 80 percent of the total CPU time. CPU utilization is determined by adding the usr and sys columns from `vmstat` (see Appendix A.22, "vmstat - Virtual Memory Management Statistics" on page 366).

Lower utilization means that the CPU can cope even better with peak workloads. CPU workloads between 85 percent to 90 percent result in queuing delays for CPU resources, which affect the response time of the application. CPU utilization above 90 percent, even for a short period, results in unacceptable response times.

While running batch jobs, backups, or loading large amounts of data, the CPU utilization can be driven to high percentages, such as to 80 to 100 percent, to maximize the throughput. This level is only achieved if the rest of the system is well tuned for these tasks too.

RDBMSs serving Web information, on the other hand, have very unpredictable workloads due to the nature of the Web and are working 24 hours a day. Having a Web site with poor performance is going to encourage customers to go elsewhere. To avoid this, we recommend sizing a Web server database to have 50 percent CPU utilization to allow for peaks in workload.

As the functionality of the CPU makes it an expensive component of a computer system, care should be taken when selecting the type and number of processors for the anticipated workload.

6.5 Memory goals and sizing

Memory is used to run programs (the code and data) for interprocess co-operation and for caching disk blocks to avoid disk I/O. For memory, there are two sizing questions:

- How much memory to have?
- How to divide this among the various uses of memory for maximum performance?

The best approach is to decide the space requirements for each memory use and add up the total. Skimping on memory can have large performance limitation consequences. Memory is used for:

- The Operating System - AIX.
- The file system cache - In AIX, this will use up any unused memory.
- The RDBMS programs.
- The RDBMS working structures.
- The RDBMS disk cache.
- The application programs.
- The users connected to the database.

6.5.1 AIX operating system

AIX operating system is a program and requires memory. AIX is dynamic and, therefore, only brings in some features when they are actually used and will grow data structures on demand. When physical memory is completely allocated, and more is demanded by programs, AIX will page out memory that has not been accessed recently to the paging space on disk. Excessive paging will hurt any UNIX system performance and must be avoided by having sufficient memory or reducing the other memory requirements.

As a usable figure, allocate 32 MB for AIX. If a graphics screen is attached to the RDBMS machine, and it is using X Windows, then add an additional 16 MB of memory.

6.5.2 AIX file system cache (AIX buffer cache)

Memory is also used by the operating system to save copies of recently used journaled file system (JFS) disk blocks. This avoids disk I/O and is, therefore, beneficial for performance. Even if the database files use raw devices, the programs will still need to use the file system cache. When the database uses the AIX file system cache, it needs a significant amount of memory.

For databases based on raw devices, allow 16 to 32 MB.

For databases based on JFS, allow 25 percent of the RDBMS cache (see 6.5.3, “RDBMS cache and structures” on page 109).

6.5.3 RDBMS cache and structures

The most important memory space used by the database is the area where the data will be read and modified, such as changing data or inserting new

rows. This memory area is called RDBMS cache. Each RDBMS product implements this cache in a different way. DB2 UDB calls it buffer pool; Oracle calls it buffer cache. Adjusting its size greatly affects the database performance. Therefore, this memory size should be as large as possible.

Memory space is also used by the RDBMS for the locking, control structures, internally used tables, areas used to control parallel queries, and saving data that reduces work, such as the SQL query plans. Making this memory space too small will drastically decrease performance.

Additional memory is necessary for database utilities, such as backup, restore, or load utilities. To achieve good performance while backing up a database, this memory size must be large enough to accommodate all the buffers that you want to allocate for this backup and for other concurrent utilities.

The effect of caching is very difficult to determine before running the system and monitoring the effect of smaller and larger cache sizes. Generally, the more memory the better, up to 2 GB, from there on, tests are required to justify that the extra memory will improve performance.

A minimum size of 64 MB should be used.

We suggest sizing initially at five percent of the raw database data size up to 2 GB in size.

6.5.4 User applications and database connections

This is the code and data of the application that each user needs to run. Because AIX uses paging space, it does not need to have the entire program in memory to run, and usually only a part of the application is required.

Only the parts of the user processes code that have actually been executed are brought into memory in the first place. If not used regularly, the code or data will be paged out to make room for other processes. As a result of this, only the parts of a process that are frequently used are held in memory. This code and data is called the process' *working set* or *resident set*. When calculating the memory requirements for RDBMS processes and applications, you need to understand that it is the resident set that is used for the calculation and not the total process size.

For example, the program on disk might be 10 MB when investigated with the `size` command. But, the resident size might be 6 MB. Also, note the resident set is made up of the code and data. The code is usually shared; so, there will only be one copy of the code in memory for all processes running this

program, but the data will be unique to each process. In our 6 MB resident size example, it might be 4 MB of code and 2 MB of data. So, for 100 processes running this 10 MB program, the memory used is *not*:

10 * 100 MB = 1000 MB <- wrong!

but is calculated as

1 * 4 MB + 100 * 2 MB = 204 MB <- correct

Note

The `ps` command can output the resident set in the RSS, TSS, and DSS columns.

Usually for applications coded in C, the amount of memory per user should be calculated at 2 to 3 MB. For more complex applications, 6 MB is a good value. If there is more than one application binary, then each must be taken into account.

For a discussion on what is really meant by *a user*, please see 6.2.3, “Sizing from user numbers” on page 105.

Sizing application programs is not simple because each of them is different. The important factors are:

- The language or environment used to implement it.
- The number of screens, features, and functions of the application.
- If the application is for general purpose (larger) or very specific (more compact).
- If the application is written for a specific RDBMS or written to work with any database (and, thus, not able to use specific features for higher performance).
- If the application is ported from an alternative OS or non-RDBMS system. Generally, these applications are large and slow.

We offer the following as very approximate starting points:

- Simple C language program - 2 to 3 MB
- Large C language program - 5MB
- 4 GL or forms - 4 to 8 MB

- Programs generated from a high-level application design tool or created within a sophisticated graphical and object oriented development environment - 6 to 15 MB

It is relatively simple to find out the memory requirements of a program from the vendor or actually measure the size of a program by running it on a test, proto-type, or any other system.

If the application is running on a machine other than the RDBMS server or the user's PC, then it does not count towards the memory requirement of the RDBMS machine. However, the application will still have to communicate with the RDBMS machine via the RDBMS server process. These are large programs themselves, but as explained above, they share the code. Allow between 4 and 8 MB per user. If the Oracle Multi Threaded Server is used, then decide how many servers you are to run instead. Each of these will be 8 MBs in size.

6.6 Disk goals and sizing

All database objects are ultimately stored on disks. These objects are the data itself, indexes, catalog/data dictionary and temporary tables. In addition, the database needs log files and rollback segments for transaction and crash recovery.

There are two levels at which to size the disks for a database:

- General, high-level, whole databases sizing
- Specific, detailed-level table by table sizing.

6.6.1 General database sizing - High-level

This has to be used when only the total size of the database is known. It has to be carefully qualified whether the size is the raw data or the disk size.

6.6.1.1 Raw data size

First, add an overhead to the raw size of the data for the placement of the data in the disk blocks of the database. Some wastage is inevitable, as one or more rows have to fit within one block; so, the last few bytes of a block are most often wasted. Databases compact the data of each row. If the column is specified with 100 bytes but only contains 25 bytes, then the database only uses 25 bytes of the block. This means more rows are stored per block. But, if an item is updated, it can get bigger and, thus, not fit within the same block anymore. In this case, the database uses a *chained block* to store the larger data item. This results in lower performance, as both blocks will need to be

read to find the row. The database allows you to specify that each block allocates some free space to cover for rows getting larger.

As a rule of thumb, most DBA use a ratio of raw data to disk size between 1:1.1 to 1:1.3. In other words, 10 to 30 percent more than the size of the data itself.

Raw data size to database size
After calculating the raw data as shown above, a scaling up calculation has to be made to cover the other parts of the database, such as indexes and working space. If there is no information available, then use the following standard raw data to disk ratios as a rule of thumb:

- OLTP - Ratio 1:3
- DSS - Ratio 1:4
- Data warehouse ratio 1:5

If you are new to databases, these values will seem high but are typical in production systems. Many people do not understand or expect the data-to-disk ratios to be so large. For example, if they use a 1:2 ratio without careful calculation, then this is unlikely to be practical, and the database will not perform well. Indeed, the data might not even be able to be loaded and indexed, or the first large query will hopelessly run out of space.
Many times people also think the index size will only be a small fraction of the raw data. For example, only allowing an extra 10 to 20 percent instead of allowing an extra 100 percent of space for the indexes (see also 6.6.2.2, “Indexes” on page 114).

6.6.1.2 Total disk size

In order to make sure that the disk size includes everything, double check if disk space for AIX, the paging space, and the database log has been included.
If in doubt, add a dedicated disk for each of the following:

- The AIX system.
- Paging if you have more then 1 GB of memory.
- Database log.

6.6.2 Specific table by table sizing - Detailed level

The alternative is the detailed level sizing of the database. Typically, this is worked out with a simple spreadsheet.

6.6.2.1 Tables

For each table, the number of rows, and then the size of one row, is estimated. By far the best way to determine this is to actually create the table in a small test database and load it with some data. A few hundred or a thousand rows will do. Then, the database can calculate the average row sizes accurately.

For DB2 UDB, use the `runstats` command, and for Oracle, use the `analyze table` command.

If the table sizes can only be estimated, refer to the database manuals for explanations on how to do this.

6.6.2.2 Indexes

Indexes can be hard to determine if this is early in the design cycle. However, the indexes might be well known from previous experience. If in doubt, guess two to three indexes per table with over a thousand rows. If the table is smaller, the indexes might not help performance and are insignificant in size anyway.

There is a minimum size of the index items for the *B tree* structure and an overhead for index structures. Most RDBMS indexes use a variant of the B tree structure to organize the index. To find a particular row, the RDBMS starts at the top of the tree and works its way down each node or branch of the tree choosing the route based on the details of the column it is looking for until it finds the reference to the row required. This final index reference is called a leaf node. For example, in Oracle, this is 22 to 25 bytes. Again, this is best worked out via a test table and index or using the explanations from the database's manuals.

The default recommendation is to assume the index size is the same as the data size. For example, if the table is 100 GB in size, then allow a further 100 GB for the index. This is based on two observations. If the table only contains a few columns (called a thin or narrow table), and only one column is indexed, then the index overhead will result in an index of roughly the same number of bytes as a row; so, the data and indexes will be roughly the same size. Alternatively, if the table has many columns (called a fat or wide table) then it is likely that many indexes will be used for the table. Each index will be smaller (as each index will only cover one column), but the multiple indexes means the indexes will be roughly the same size as the table.

6.6.2.3 Temporary space (sort space)

This is used for DBA activities, such as creating indexes, creating summary tables, loading data, and for very large SQL query results to be stored and sorted before being returned to the application and user.

To index a large table requires a large amount of temporary space. It is not precise, but the space to allow for a full table sort on disk can be up to 1.3 times the size of the table. If the table is indexed in parallel, this can take up to two times the table size. Most databases have one dominant table. If this table is 30 percent of the data size, then the temporary space needs to be 60 percent of the data size. This assumes that nothing else is using temporary space while indexing.

For DSS databases, there is a large need for temporary space due to the large numbers of rows being sorted and the summary table creation.

Most systems use the rule of thumb of having the same amount of temporary space as data space. Running out of temporary space results in a complete failure to create indexes, which makes performance impossible or SQL statements failing with errors. Both must be avoided at all costs.

6.6.3 Which disk size to choose

Currently, there are 4.5 GB, 9.1 GB, 18.2 GB, and shortly, 36 GB disks drives available for the RS/6000. There are several trends in disk technology:

- They get bigger every year, roughly following Moore's law, which states that computer power doubles every 18 months.
- The cost per GB is lower each year.
- The cost difference of the two smallest drives diminishes until there is little point in continuing with the smaller drive.
- The disk drives improve a little each year in seek time.
- The disk drives get smaller in physical size.

All this means is that the databases use disk drives that are bigger in space size and smaller in physical size. The speed improvements are, however, small in comparison.

This means that a database that would have taken 36 * 1 GB drives four years ago can now be placed on one disk. This highlights the database I/O problems. For example, if each 1 GB can do 80 I/O operations a second, this means the system can do a combined 36 * 80 = 2880 I/O operations per second. But a single 36 GB drive with a seek time of 7 ms can do 140 I/O

operations per second. Clearly, the new disk drive capacity is good news, but lower numbers of disks cannot deliver the same I/O throughput.

The only way to work out the I/O throughput requirements is to:

- Use some test or proto-type system from which to measure the I/O for a given workload.
- Determine the transaction rate and read and write operations per transaction for OLTP systems. Remember reading data will involve reading indexes, and inserts and updates require data, index, and logs to be written.
- Estimate the number of rows and tables that will be scanned for typical query types for DSS systems.

The writers of this redbook recommend using the smallest drive possible purely on the basis of increasing the number of disks for I/O throughput. The alternative (when the two smallest drives are nearly the same price) is to buy the next largest drive and only use half the disk space. The middle area of the disk is the fastest; so, it would make sense to use this. This leaves the other half of the disk available for other things, such as:

- Disk to disk backup
- Archiving data
- Test databases for out-of-hour testing
- Extra copy of the database for upgrade testing

Note

Traditionally, we decide the number of disks by dividing disk space requirement by the current disk size, but as disk drives increase in capacity, it will become more important to decide the number of disks via the disk I/O requirements.

6.6.4 Disk protection

Disks crash. We have to live with this and build a system that can tolerate this problem. The system can (given sufficient funds) be built to carry on running with zero interruption or to be recovered in a few hours, despite a crash.

The database log needs disk protection to allow database recovery of recent transactions. The other disks can optionally be protected. If they are protected, downtime while data is recovered can be eliminated.

In practice, there are two options:

- RAID 5
- Mirrored disks - With PP striping or fine striping from AIX 4.3.3 onwards

See Chapter 8, “Designing a disk subsystem” on page 149 for more information on disk options and performance. In terms of sizing, both options mean more disks and suitable adapters.

- RAID 5 - We recommend using a seven data to one parity disk ratio; so, add one seventh to the number of disks.
- Mirror - Simply double the disk requirements, but do not forget you may need additional adapters for the disks.
- You may choose to implement a mixture like mirrored log disks and RAID 5 for data, index and temporary space (but note temporary space is 50 percent read and 50 percent write).

Do not forget the AIX and paging space disks. A failure on these disks can bring down your system. If a paging space disk fails, the system will restart without it, but if the AIX disk fails, it can take extra long to correct it. The AIX `mksysb` backup method will minimize this down time. The alternative is mirroring these disks - RAID 5 is not recommended.

Also, do not forget to include a spare disk or two for speeding up recovery and the risks while running after a disk failure.

6.6.4.1 Minimum disk requirements for small databases

Table 4 highlights that, with small database and small numbers of disks, you have to be very careful with the 1:3 rule of thumb, as it only applies to larger databases and higher numbers of disks.

Table 4 gives some example database sizes (assuming 4.5 GB disks).

Table 4. Example database sizes:

Use number of disks	Absolute minimum disks	Small RDBMS	Small and safe RDBMS	Large RDBMS
AIX	1	1	1 + mirror	1
Paging and RDBMS code	use above	1	1 + mirror	2
RDBMS data	1	1	1 + mirror	8
RDBMS indexes	1	1	1 + mirror	8

Use number of disks	Absolute minimum disks	Small RDBMS	Small and safe RDBMS	Large RDBMS
RDBMS temp	use above	1	1 + mirror	8
RDBMS logs	1 + mirror	1 + mirror	1 + mirror	1
Database data	2 GB	4 GB	4 GB	36 GB
Total disk size	22 GB	31 GB	58 GB	128 GB
No. of disks	5	7	13	28
Data to Disk Ration	1:11	1:7	1:14 or 1:7 with mirror	1:3.5

In the *Absolute minimum disks* column, we have allocated the index and temp space onto one disk. This is not ideal, but might work in practice because databases tend to use indexes for transactions or temp space for index creation and sorting full table scan large queries, but not both at one time. This column highlights the minimum number of disks for an RDBMS. This is not a recommended minimum disk subsystem for a database but does have the lowest cost.

The *Small RDBMS* column is a recommended minimum disk subsystem although there may be limits in I/O rates due to the data being placed on only one disk. Striping the data, indexes, and temp across these three disks might help reduce this. This does not include disk protection for the database or other disks (apart from the mandatory log disk protection for transaction recovery).

The *Small and safe RDBMS* column adds full disk protection and would survive any disk crash with zero down time.

The *Large RDBMS column* highlights a normal sized database and approaches the 1:3 ratio rule of thumb. We could add disk protection to this configuration too.

Summary

Use the simple 1:3 ratio rule with a minimum number of disks to decide the database size and number of disks and then add disk protection.

6.7 Balancing a system via the component *costs*

Once a machine size has been determined, it is worth making a few checks to make sure that the system is sensible and will work in practice. The teams that size database systems regularly have found that the majority of configurations have a constant ratio between the power of the CPU, the memory, and the number and size of disks. This is based on the idea that a certain CPU power running an RDBMS will generate a certain level of disk I/O to supply new data for processing. The memory is then related to the database size and reduces the Disk I/O by caching data.

Rather than giving you a lot of ratios and calculations to work out for each machine in the range, Table 5 gives you the percentages of the cost of the machine for the main components.

Table 5. Percentage of cost for CPU, memory and disks

System Size	Example	Percentage of cost		
		CPU	Memory	Disk
Low End	43P	40	10	50
Mid Range	F Series or H Series	40	20	40
High End	S Series	40	25	35

With this information, do not forget that, in addition, you will probably need:

- Network adapters
- Backup the system to tape drives
- Software

This also points out that on smaller machines the memory is relatively inexpensive, but the disks are not because you need a minimum set of disks to run a database.

Other things to notice are:

- CPU - On SMP machines with less than the full number of CPUs, it is a very simple and inexpensive task to upgrade. But, machines with the full number of CPUs are less expensive in cost per CPU power terms.
- The upgrade from machines with one CPU or the maximum SMP CPUs is often not simple. The next machine up in the range might be quite different and require the CPU, motherboard, complete cabinet, memory, and adapters changed. This means the upgrade will take much longer in downtime and will be more complex. Also, larger machines cost more. A

requirement for a ten percent improvement in CPU terms might mean actually having to install a machine that is 50 percent faster and much more costly.

- Memory - Most machines are not initially configured or installed with the maximum memory in the system. This means extra memory can simply be added to the machine if the initial size proves to be too small.
- Disks - Compared to the cost of the machine, the cost of a single disk drive is very small. Adding external disks does not have to involve the main system cabinet. SSA disks, in particular, can be very simply added to the system with zero downtime. Even adding a disk adapter is a simple process. Disk space is also the first part of the system that is likely to outgrow the initial size.

Summary

Sizing is often choosing the RS/6000 model based on the CPU power rating and then balancing memory and disks so that full use of the CPU(s) can be achieved. Use the ratios above to double check your sizing is balanced.

For more sizing information refer to the redbook *Understanding IBM RS/6000 Performance and Sizing*, SG24-4810.

Chapter 7. Designing a system for an RDBMS

When designing a system for implementing an RDBMS, you should be aware that the deeper you comprehend all the basic design concepts necessary to build a system, the better the design and the more stable the system will be.

Most of the design considerations discussed in this chapter can not only affect the AIX and RDBMS performance but also the availability of your whole system. All the fundamental considerations about physical space, memory usage, and CPU consumption must be analyzed prior to implementing a system, and they must also be planned for future needs. This chapter will also discuss the different machine groups as well as explain how to maintain the database's security through the use of backups.

7.1 Working space

When you are designing an RDBMS, you have to keep in mind that there are basically three vital concerns for a well-designed system:

- Basic and future AIX resources
- Basic and future application resources
- Basic and future RDBMS resources

7.1.1 Basic and future AIX resources

The first point to be considered is how much space the AIX operating system will consume when installed on an RS/6000. The current AIX Version 4.3.3 needs approximately 400 MB of disk space for basic installation and graphical tools. Besides this basic disk space allocation, the system administrator will have to increase the AIX paging space size using the value suggested by the AIX Installation Assistant tool. The real indicator as to whether the paging space might be increased is the database's resource consumption as requested by users and applications. Once the system is implemented and the number of user and application requests increase, it is possible to monitor the paging space consumption in order to determine if it is necessary to define more space for it. As a rule of thumb, it is recommended that the minimum size for the paging space area is set to the same size as real memory. Sometimes this value has to be increased to two times the size of the machine's real memory. If more than 1GB will be used for paging space, and if more than one disk exists in the machine, it is recommended to split the paging space across the disks usually using 1 GB per disk drive. You should be aware that whenever you use the paging space area, the overall performance automatically decreases. Please refer to Chapter 10,

"Monitoring an RDBMS system for performance" on page 203 for further information about monitoring tools and techniques.

Keep in mind that new AIX features and fixes can eventually consume more space than what was initially designed to be available for AIX. Although these changes do usually not consume a substantial amount of disk space, they still must be considered as part of the future AIX resource allocation.

At the time AIX is installed, it is recommended to load most of the frequently used features, such as the online manuals, in order to avoid future space consumption and unnecessary workload. Please consider 2 GB of disk space for all the AIX Journaled File Systems, such as /tmp, /home, and /var.

7.1.2 Basic and future application resources

Production tools, third-party tools, and applications used on a system must be considered when planning for disk space. These kinds of products are subject to be replaced and/or added constantly, especially due to the constant application development cycle. The initial needed space and long-term growth must be planned in the system. Please refer to the application vendors in order to find out about the space requirements for their particular products.

7.1.3 Basic RDBMS resources

This is the core area of an RDBMS system, and errors in the design can result in major performance problems. Therefore, special care should be taken when designing the resources for the RDBMS.

7.1.3.1 Basic DB2 UDB resources

When designing a system for a DB2 UDB RDBMS, the following physical files have to be taken into consideration:

Log files

The log files contain all the information regarding the database changes caused by an `INSERT`, `UPDATE`, `DELETE`, `CREATE`, `ALTER`, or `DROP` command. They are used in a database failure situation or even in a system crash for restoring the database to its consistent state. This action ensures the integrity of the database.

When the *circular logging* is being used, the only thing to be aware of is the possible secondary log file's disk space allocation. When the *archival logging* is being used, a user-exit must be coded for moving the archived log files to a tape device, thus, freeing the disk space. Please refer to 4.1.3, "Physical storage structures" on page 59 for further information about the different types of logging.

Data files
Data files are the main area to be considered when designing an RDBMS system. This is the place where the database stores its data, and space consumption is usually quite high, independent of whether SMS or DMS tablespaces are used. Because this is such a dynamic area, system and database administrators need to monitor the growth of the data files very closely. For information about the different types of tablespaces, please refer to 4.1.3.3, “Data files, index files, and temporary space” on page 61.

Index files
As in the Oracle RDBMS, the indexes will be defined for a table, and they will be responsible for speeding up some queries. Since they consume some disk space, it is necessary to estimate the disk space consumption size. Please refer to Chapter 3, “Designing Your Physical Database - Index Space” in the manual *IBM DB2 UDB Administration Guide: Design and Implementation, Version 6,* SC09-2839, for a complete equation description.

DB2 UDB control files
Each DB2 UDB database has its own set of control files used for managing the database’s internal functionality. These are:

- SQLDBCON
- SQLOGCTL.LFH
- SQLINSLK
- SQLTMPLK
- SQLSPCS.1
- SQLSPCS.2
- SQLBP.1
- SQLBP.2
- DB2RHIST.ASC
- DB2RHIST.BAK

These files should not be deleted, renamed, or moved from their original location. The RDBMS will be responsible for saving them to a file or a tape device when a database backup is taken. These files do not represent more than 1 MB of disk space.

Sort space
Some SQL statements require that the RDBMS creates some temporary tables in order to process them. This is the case where an `ORDER BY` statement is used on a large amount of data, therefore, causing a sort operation to occur. If the amount of data is bigger than the size of the available memory, a temporary table will be created on disk in order to execute the order

operation. Both RDBMSs will create the temporary tables on the physical space that was defined for the temporary tablespace.

Two rules of thumb can be used for sizing the sort space:

- Make the sort area size 1.5 times larger than the largest table of the database. This can be used when the database has very few big tables and a lot of small tables.
- Make the sort area size equal to the sum size of all tables on the database. This can be used when all the tables have approximately the same size.

Since the amount of required space will be totally dependent on the queries and the amount of data returned for the sort operation, the only way to figure out the ideal size for the temporary tablespace will be through the monitoring process once the database is operational.

Reorganization space
Reorganization is necessary to remove the wasted space caused by deleted rows. Whenever you need to reorganize a table or the indexes that are defined on it, you will need some staging area to be allocated.

If the reorganization takes place on DB2 UDB, the temporary tablespace will have to be increased in size, or maybe another staging tablespace can be created for the reorganization step.

If the reorganization takes place on Oracle, another staging table, with the same size of the table to be reorganized, must be created in order to allow the data to be transferred in and out of the table. Another option is to export the table to a file on a disk, drop the contents of the table, and import the data again.

Please refer to 7.7, “Coping with growth” on page 137 for more information about table and index reorganization.

7.1.3.2 Basic Oracle resources
Oracle RDBMS needs disk space for the following physical files:

Redo log files
The redo log file records all changes made to the user or system objects.

They are also used for error recovery in case of media failure or a system crash. Each Oracle instance requires at least two redo log files, but this number can be expanded for availability or security reasons. The copies

should also reside on different disks if there are disks available on the system.

The more INSERTS, UPDATES, DELETES, CREATES, ALTERS, and DROPS you have in your database, the more physical space for the log files should be planned. Once the database determines that all the transactions recorded in the log file were committed, that is, when the database puts the log files in a state called *archived*, the log files are ready to be moved out from disk to a tape device, thus, freeing the disk space. Due to this very frequent update characteristic, the redo log files should be put on dedicated disks when possible. As a rule of thumb, consider 4.5 GB to 9 GB of disk space for the redo log file.

Data files
This is the main area that both system and database administrators should be concentrating on because it is usually a source of never-ending space consumption, and the database must have this space available for storing the new added rows.

Index files
Whenever you want to avoid sorts, speed up frequently executed queries, and provide some organization in the table, index usage should be considered. However, as with the data files, the index files sometimes consume large portions of disk space depending on the number of indexes defined for each table and the size of the table that you want to index. For an estimated amount of disk space consumed by an index, please refer to “Appendix A Space Estimation for Schema Objects” in the *Oracle8 Administrator’s Guide, Release 8.0*, A58397-01.

Control files
The control files include information about the database itself. They are small in size, usually less than 1 MB, but crucial for starting the database. Without them, the database cannot operate properly.

Initialization file
The initialization file is usually named *init.ora*. This file is read at the database startup time, and it can contain more than two hundred parameters that can influence the performance and functionality of the Oracle RDBMS. This is a text file, and its size is usually less than 20 KB.

Rollback segments
A rollback segment is responsible for recording the old values of data that were changed by each transaction and is used to provide read consistency to roll back transactions and to recover the database. Compared to the size of

the data files, a small amount of space should be assigned to the rollback segments. Please refer to *Oracle8 Administrator's Guide, Release 8.0, Identifying Rollback Segment Contention*, A58397-01, for more information about the size of the rollback segments.

7.1.3.3 Backup space

A backup copy is necessary for the whole machine, for AIX (command `mksysb`) but also for each existing database. However, the backup strategy will depend on the size of the database and also on the free disk space available.

For small and medium size databases, allocating disk space for the database backup copy might be a reasonable strategy. However, for large databases, this concept tends to be impractical due to the huge amount of disk space you should have available for each different database backup copy and for each different daily, weekly, or monthly backup strategy. For this scenario, it is recommended that you back up your database directly to an external unit, such as a tape device.

If you chose to generate and store database backup copies on disk, you should use separate disks for the database files and the backup copies. This can avoid hardware problems destroying the database and all the backup copies at the same time.

Please refer to 2.7, "How do we make the data safe?" on page 31 for a general description on backup strategies.

7.1.4 Future RDBMS resources

When designing a system, it is important to first consider the space needed for an immediate start. Then, you must have a clear idea of how much disk space is needed for:

- The start point
- The database growth for the *near future*
- An estimated database growth for the *remote future*

The discussion of what can be considered near future and remote future will depend on the database growth rate. For a read-only database, the near future can be estimated to be six months, and maybe the remote future one year. However, on a OLTP environment, one month can be usually considered as the near future, and three months can be considered to be remote future. These are only suggested value since each company will have its own growth rate, thus, changing the time frames for what is considered near and remote future.

Some space should be planned for future RDBMS upgrades and additions of new features.

Please refer to 7.7, “Coping with growth” on page 137 for more information about the most commonly changing RDBMS areas.

7.1.4.1 Overall disk space allocation

The disk space has to be planned for all areas, such as initial and future needs, in order to avoid possible lack of space situations. Some areas, however, will demand special attention due to their expected growth rate.

Each customer’s disk space consumption areas will vary depending on the workload characteristics. Table 6 describes the different disk space consumption areas and reports what is commonly expected for DB2 UDB and Oracle.

Table 6. Growth rates for planning disk space

Disk space consumption area	Expected growth
AIX (updates and upgrades)	Small
Application (languages, production, and third-party tools)	Medium
Oracle archived REDO log files	High
Oracle data files	High
Oracle index files *	High
Oracle control files	Small
Oracle initialization files	Small
DB2 UDB log files	High
DB2 UDB data files	High
DB2 UDB index files *	High
DB2 UDB internal control files	Small
Temporary tablespace for sort operation	Medium
Backup space	Medium
* The growth of the index files will depend on how many indexes are defined for a table and the table’s growth.	

7.2 Workload considerations

Before you start, it is necessary to have a clear idea of how the database will be accessed. In other words, you need to know the database's workload characteristics. For example, a database that is used to store patients' personal information in an emergency room is completely different to a database holding historical data in a museum. The first one has a unique characteristic of inserting data, while the data stored for a museum is basically queried during the whole day.

Please refer to Chapter 3, "Types of workload" on page 43 for a better understanding of Workload characteristics.

7.3 Network considerations

One thing to be considered when designing the system is how the network can influence your overall performance.

Usually, there are very few local connections to the server machine. In the majority of the cases, the user applications run on client workstations that send and receive requests through the network. Even if the database is able to process the requests immediately after they arrive, a serious performance issue will exist if there is a network delay in the following situations:

- The time between when a client machine sends a request to the server and the server receives this request.
- The time between when the server machine sends data back to the client machine and the client machine receives the data.

Once a system is implemented, the network should be monitored in order to assure that its bandwidth is not being consumed more than 50 percent.

There are two application techniques that improve overall performance and avoid high-network consumption:

- Transmit a block of rows to the client machine in a single operation. When using DB2 UDB, this can be accomplished by using the `BLOCKING` option in the pre-compile or bind procedures. Please refer to the *IBM DB2 Universal Database Administrator Guide: Performance Version 6*, SC09-2840, for further information about row blocking. When using Oracle, this can be achieved by a process called *extracting data via arrays*.
- Use stored procedures to minimize the number of accesses to the database. Stored procedures are programs that reside on the server side

and can be executed as part of a transaction by the client applications. This way, several pre-programmed procedures can be executed by using only one `CALL` command from the client machine. The stored procedures can be coded in different languages depending on the RDBMS. For Oracle, the PL/SQL or JAVA extensions can be used. For DB2 UDB, there is a choice of Java, C, Cobol, or SQL procedures.

Besides the application enhancements that can be provided by DB2 UDB and Oracle, it is recommended that medium and large databases are connected to the client machines using FDDI or 100 MB Ethernet LANs. A 10 MB Ethernet LAN usually only provides an acceptable bandwidth for small databases.

7.4 Memory and database considerations

The database administrator will be responsible for balancing the database's memory usage, while the system administrator will balance the overall memory usage. One of the most ordinary cases, where both areas should work together, is where an RDBMS and an application, such as SAP, have to share the same machine. A practical example is that both of them have their own *buffer pool* for treating the data, and the memory space dedicated to each application should be carefully chosen.

The RDBMSs use memory to manipulate their own data in order to satisfy the customer's requests. The amount of memory to be planned for an RDBMS is proportional to the number of users and applications that will be connecting to the database. The memory space needs are different for Oracle and DB2 UDB.

7.4.1 DB2 UDB memory requirements

The following table suggests the amount of memory that is required to run DB2 UDB based on the total number of possible connections. The real memory need will vary depending on the functions that the users and applications are using.

Table 7. Memory requirements for DB2 Universal Database

Number of clients connecting to a server	Memory space needed
10 concurrent connections	80 MB
25 concurrent connections	96 MB
50 concurrent connections	186 MB

Number of clients connecting to a server	Memory space needed
DB2 UDB Administration Tools	30 MB

For example, if the system will have the DB2 UDB administration tools and 65 users connected to it, it is necessary to provide 30 MB for the administration tools, 186 MB for 50 users, and 96 MB for 25 users. The total suggested memory is 312 MB.

Whenever there is an increase in the number of users and applications connected, the memory size should be recalculated.

DB2 UDB has several parameters that affect memory consumption, either on the server side, client side, or on both. Please refer to 12.4, “What are the options?” on page 265 for further explanation about the different types of memory allocated by DB2 UDB.

7.4.2 Oracle memory requirements

The first thing to be calculated for an Oracle RDBMS is the size of the Shared Global Area (SGA). This area contains all the space necessary for the Database Buffer Cache, Redo Log Buffer, and Shared Pool. Please refer to 2.5.1, “RDBMS terms” on page 22 for a further information about the SGA.

The following initialization parameters control the size of the Oracle Shared Global Area:

- DB_BLOCK_BUFFERS
- DB_BLOCK_SIZE
- SORT_AREA_SIZE
- SHARED_POOL_SIZE

These parameters should be set with caution since a too high value for them could lead to exhaustive memory usage and paging. The sum of all instances’ SGA sizes should not exceed more than fifty percent of the total memory.

The approximate size of an instance's SGA can be calculated with the following formula:

```
(DB_BLOCK_BUFFERS * DB_BLOCK_SIZE)
+ SORT_AREA_SIZE
+ SHARED_POOL_SIZE
+ 1 MB
```

After you have defined the SGA size, and prior to starting Oracle RDBMS, the following formula can be used to estimate the total memory requirement for the server:

```
Size of the Oracle executables
+ size of the SGA
+ number of background processes * size of tool executable's private data
section
+ size of the Oracle executable's un-initialized data section
+ 8192 bytes for the stack
+ 2048 bytes for the processes' user area
```

The command `size -f` can be used to retrieve an executable's text size, private data section size, and uninitialized data section size.

Oracle also calculates memory space allocation based on the number of users and applications that will connect to the database.

For each client connection, the following formula can be used for estimating the memory consumption:

```
Size of the Oracle executable's data section
+ size of the Oracle executable's un-initialized data section
+ 8192 bytes for the stack
+ 2048 bytes for processes user area
+ cursor area needed for the application
```

Whenever there is an increase in the number of users and applications connected, the memory size should be recalculated.

Please refer to the *Oracle8 Server Tuning Manual*, A54638-01, for more information about the Oracle memory tuning.

7.5 System resource utilization

An RDBMS should run on a machine that fulfills all the system resource demands, from the operating system needs through the connected applications, asking for a row from the database. Prior to choosing the most appropriate machine type and model, it is essential that some basic points are clearly defined, such as the amount of users and applications that will

access the database, how much CPU, memory, and disk the RDBMS needs for handling its own tasks, and the system workload characteristics.

Once the total amount of resources are defined, it is possible to build the databases and monitor and tune them in order to achieve the machine's maximum planned capacity usage.

The CPU, memory, and disk resources must be well planned in order to speed up the system through the use of their maximum capacity but always avoid to overcommitting them.

The expected CPU consumption will depend on which system workload is running on the machine as follows:

- OLTP
  70 - 80 percent
- Web
  50 percent
- DSS
  80 - 90 percent
- Batch
  90 - 100 percent

On an SMP or SP machine, this workload must be split evenly all over the CPU's. If the CPU utilization is higher than the recommended values over an extended period of time, this might be an indicator for a possible CPU bottleneck.

Operating system and RDBMS will allocate memory in order to perform the operations requested by users and applications. The available memory resources should be completely used most of the time, but care should be taken that no paging occurs on the system due to a bad memory consumption plan.

The disk subsystem design should allow the database's data to be evenly distributed among the disks. The data placement step must be very well designed, thus, avoiding data skew. A bad data placement can easily lead to a bad I/O request distribution, that is, some of the disks can be continuously used, while others can be available and without any task to perform. The more even the distribution is, the more the performance gains that can be achieved. When all the disks are being accessed evenly, the disk utilization rate for each disk should be around 40 to 50 percent.

These three resource consumptions should always be monitored and under control, therefore, assuring that the RDBMS is able to completely and evenly explore all the available resources. For more information about the monitoring process, please refer to Chapter 10, “Monitoring an RDBMS system for performance” on page 203.

7.6 Can the database be backed up and restored?

Maintaining a backup and restore strategy is not only a good practice but a real necessity.

Backing up the database allows recovery of the database either partially or totally in case of an operating system, RDBMS software, or hardware failure. These can damage or make the database inoperative and the data inaccessible.

Each company has a different need and a different approach for the backup strategy. Some read-only environments, such as a DSS, will keep two or three backup images but will not be interested in taking frequent backups as an OLTP environment should do.

The backup and recovery type, either totally or partially, will depend on how the logging mode (for DB2 UDB) and archive mode (for Oracle) are set. Both RDBMSs, although using different terms, have the same ability to configure the two possible database restore scenarios:

- Partially
  After the database is restored from a backup image, it will only allow users to access the data available at the time the backup image was taken. All the database changes made from that point on are lost.
- Totally
  After the database is restored from a backup image, all the transactions made to the database will be reapplied, thus, allowing users to access the last committed transaction prior to the crash. No database change is lost.

7.6.1 DB2 UDB backup/restore scenario

In order to make a backup copy of a database, the `db2 backup` command is issued. This command can be issued with two different options: off-line and online.

The off-line backup is mandatory when the database parameter LOGRETAIN is set to NO, which is also known as circular logging. It indicates that no connections to the database can exist at the time the backup is taken.

If the LOGRETAIN parameter is set to RECOVERY, also known as *log retention logging*, the backup can be taken with all users and applications connected. It can be taken for either the whole database or only for some tablespaces. When this backup is taken, the database administrator should always be aware that the future *archived logs* (logs that contain all units of work that have been committed) must also be copied to a safe unit since they store the data needed to update the database to the last committed transaction in case a crash occurs. This is the most recommended backup strategy for a 24x7, non-stop system.

Now suppose that your database became inconsistent due to an external factor, such as a power failure, media problem, or application failure. When a new connection is made, or when the `restart` command is issued, it initiates an activity called *crash recovery.* Crash recovery consists of rolling back incomplete units of work from the time a failure took place, thus, allowing the database to be operational again. The RDBMS uses the database log files for the database crash recovery process.

If the database cannot be put into a consistent state again, for example, in cases where the log files are also damaged, one of the two recovery methods should be used: *Version recovery* or *roll forward recovery*.

- Version recovery uses off-line backups for recovering the database to a consistent point again. The database can only be restored off-line, and it is restored to the same state it was in when the off-line backup operation took place.
- Roll-forward recovery also uses off-line backups for recovering the database to a consistent point again. Apart from that, roll forward recovery has the possibility to apply all the changes made to the database from the last time the backup was taken to the last committed transaction. This is done by reading and applying all the database changes stored on the active and archived log files. This process is called *roll forward*.

7.6.2 Oracle backup/restore scenario

When the database is abruptly interrupted by an external factor and must be set into an operational state in order to allow connections again, an *instance recovery* will take place. The online redo logs are used to roll back transactions that were not committed at the time the database had the problem.

If the online redo logs are also damaged, Oracle will provide two different ways of restoring the database depending on which archive mode is in use: *ARCHIVELOG* or *NOARCHIVELOG*.

When the NOARCHIVELOG mode, also called *Media Recovery Disabled*, is used, the archiving of the online redo log is disabled, and a recovery will only be possible up to the point the last off-line backup was taken. This mode does not protect the database from a possible media failure since it only allows the database administrator to take off-line backups.

If the ARCHIVELOG mode is used, archiving of the online redo log is enabled allowing the database to be restored to the last committed transaction. This is done through a process called *roll forward*, which consists of applying the redo logs to datafiles and control files. This mode also allows the database administrator to take off-line or online backups from both the whole database as well as from certain tablespaces only.

7.6.3 General backup considerations

For security reasons, the backup/restore routines must be treated as a two-step procedure. Only backing up a database and never testing the restore might have the same catastrophic results as having no backups at all. Although the backup utility for both RDBMSs are very functional and stable, the restoring procedure is as important as the backup on a controlled system.

The restoring test should not be done after each backup but should happen periodically on a scheduled day, preferably on another machine. This can help to determine the amount of time required for recovering the database.

Depending on your business requirements and on the size of the database being backed up, different backup/restore approaches should be considered. The next two examples can show practical approaches for the different possible backup methods:

- Example 1 - Using Tablespace Backup
  In a database where only some specific tables are being updated, and they represent a small percentage of the whole database, it is desirable to distribute them among separate tablespaces and back up only these tablespaces. Besides being faster, it might also save some disk space.
- Example 2 - Using Backup Online
  For a 24x7 system, the best option is the use of online backups, which allow users and applications to be connected while the backup is running. Although it might cause a little overall performance decrease, it is the best way for assuring a secure database backup without closing the transactions or disconnecting users and applications.

These are the recommended backup procedures for the different database scenarios:

Database can be stopped daily

- Daily - Take an off-line backup
- Monthly - Choose one of the daily backups as a permanent monthly archive
- Yearly - Test the integrity of the backup archives

Database can be stopped once a week

- Daily - Back up the log files, back up the most important tables and take an online backup
- Weekly - Take an off-line backup
- 3 month - Choose one of the weekly backups as a 3 month archive
- 6 month - Test the integrity of the backup archives

Database cannot be stopped

- Daily backup. Choose one of the two following options:
  - Full online backup of the database including the log files.
  - If the database is large, then make an online backup of the log files, the important volatile tables (that have a lot of inserts and updates but are not too large), and 20 percent of the database. This means that over five days the entire database is backed up and the backup data volume is reduced.
- 3 month - Choose one of the backup sets (possibly five days worth) as a 3 month archive.
- 6 month - Test the integrity of the backup archives set (five days worth).

Note

Always store the backup media off-site!
Consider, however, that backups stored on tape are often in plain ASCII. Therefore, sensitive data should be encrypted, which might increase backup time but will protect the data when it goes off-site.

Since the backup and restore processes represent a very demanding task, it is recommended that some external tools are used in conjunction with the RDBMSs. These tools must be able to ease the data security management by automating the backup and restore processes. A recommended storage management solution, that can be used with both RDBMSs, is the Tivoli Storage Management ADSTAR Distributed Storage Manager for AIX.

Many customers, usually when working in some big environments, tend to have the backup functionality residing on another machine, usually called a *backup server machine*. When this is the scenario, where the backup will take place, it is recommended that another dedicated physical network is used for data transmission between the RDBMS production server and the backup server. This can avoid users and applications being impacted by an overall network performance degradation caused by the huge amount of data transmitted between the servers when the backup is started.

7.7 Coping with growth

Coping with database growth is a task that demands close and planned monitoring; this is essential for having a clear idea on how the database is growing day by day. The database growth involves the following areas:

- Increasing number of tables
- Increasing number of indexes in each table
- Increasing physical space for the table data
- Increasing physical space for the index data
- Increasing number of connected users for each database
- Increasing number of applications accessing each database

The total sum of all these factors could lead to an increase usage of CPU, memory, disk, and network resources. Among all of these factors, the most delicate area is the lack of disk resource.

When you do not have a clear idea of the disk allocation for immediate and future needs, you could find yourself dealing with a data fragmentation problem in the future, especially when data is split among several files on the same disk. When the data is not stored on raw devices, it is recommended that the data is split evenly across all disks, thus, increasing the performance by making the data parallel accessible. Data is stored on tablespaces, and tablespaces map to physical files or disks. It is recommended that each tablespace file is created on a separate disk in order to spread out disk I/O.

Whichever RDBMS you are working with, data and index reorganization represents an important task. Extra temporary space will possibly be needed. The frequency of the reorganization, however, will depend on how the tables and indexes were created, the frequency with which the tables are updated and how long users can remain without accessing the tables during the reorganization phase. This only applies to Oracle since DB2 UDB is able to reorganize the table with users connected to the database.

7.7.1 DB2 UDB reorganization method

When coping with DB2 UDB RDBMS growth, you should keep in mind that the tables and indexes might need to be reorganized and that the system catalog tables are always up-to-date to reflect database growth. An easy way to determine if a table or an index (or maybe both) must be reorganized is to run the `reorgchk` command.

When checking the table data organization, the `reorgchk` command will display an entry called CLUSTERRATIO, which indicates the percentage of table data that is organized according to an index. If this value is less than 80, the table needs to be reorganized according to the most used index. When a table has multiple indexes, some of them will always be in a different sequence than the table, but this is expected. However, you have to specify the most important index for reorganizing the data.

For checking the organization of the indexes, the `reorgchk` command will display an entry called 100*NPAGES/FPAGES, which indicates how organized an internal index page is. If this value is less than 80, the indexes must be dropped and re-created.

DB2 UDB provides the facility to perform an automatic index reorganization online without the need to force the connected users and applications off the database. Special attention must be paid to the size of the temporary tablespace since this is the space that is used for the reorganization procedure by the database.

7.7.2 Oracle reorganization method

When coping with Oracle RDBMS growth, the simple method is using the export and import routines frequently in order to keep the data organized.

To check the table data organization, the command `ANALYZE TABLE` *`table_name`* `COMPUTE STATISTICS` must be used. This command collects storage statistics that can be queried through the following statement:

```
SELECT NUM_ROWS, BLOCKS, EMPTY_BLOCKS, AVG_SPACE, CHAIN_CNT, AVG_ROW_LEN
FROM ALL_TABLES
WHERE TABLE_NAME = 'table_name'
```

If you have many empty blocks (column `EMPTY_BLOCKS`) or many migrated rows (column `CHAIN_CNT`), it is recommended that you reorganize the table data. This can be accomplished by exporting the data to a file or to another table, therefore, dropping the original table and inserting all the rows again.

In order to check the organization of the indexes, the command `ANALYZE INDEX index_name COMPUTE STATISTCS` must be used. This command collects statistics from all the indexes dictionary views and stores them in the BLEVEL column. Once the statistics are collected, the following command can be run in order to analyze the reorganization need:

```
select index_name, blevel from all_indexes where index_name='index_name'
```

If the value of the BLEVEL column is greater than four, the index must be rebuilt. This can be done by issuing the `ALTER INDEX index_name REBUILD` command.

Only the index can be reorganized online if Oracle is used on a 24x7 system. In order to reorganize the data on a table, the users will not be able to access that specific table while the reorganization is taking place. Besides, it is important to allocate a staging space, either on a table or on a disk, for holding the data of the table that is being reorganized. This alternative can be accomplished by creating a new table using the Oracle command `create new_table as select * from old_table`, dropping the old table, and renaming the new table to the name of the old table. Although possible, this is an alternative that demands available disk space.

7.7.3 When and how to avoid database reorganization

Although the reorganization procedure usually provides a good performance increase, it is also an expensive task. Databases residing on a 24x7 system usually cannot afford to have inaccessible tables during reorganization (as in Oracle), or they might not be affected by the performance decrease caused by an online reorganization (as in DB2 UDB).

For this situation, both RDBMSs suggest the use of a parameter called *PCTFREE*, which indicates the percentage of a data blocks (Oracle) or data page/index pages (DB2 UDB), to be reserved as free space for future updates and inserts.

The following sections offer suggestions as to how to avoid reorganization.

7.7.3.1 Avoiding DB2 UDB reorganization

Since DB2 UDB implements PCTFREE parameters for both data and index pages, the following method can be used, after creating the table structure, in order to avoid reorganization:

1. Alter table to add PCTFREE
2. Create clustering index with PCTFREE on index

3. Sort the table data externally
4. Load the data

The recommended value for PCTFREE will depend on how frequently data is updated or inserted into the table. The PCTFREE value for indexes is set to 10 by default, and it is a good initial value. For tables, it is a good practice to define the same value and use the `reorgchk` command to monitor how long this PCTFREE value helped to keep the data organized. Depending on the output, this value can be increased or decreased. In order to reduce the frequency of dropping and re-creating the indexes for reorganization, the parameter *MINPCTUSED* can also be defined when creating an index. This parameter specifies the threshold for the minimum amount of used space on the indexes leaf page. This space is automatically reclaimed after a DELETE operation if this threshold is reached. The default value for MINPCTUSED is zero, which means that online reorganization is disabled. The recommended value should be less than 50 percent in order to merge two neighboring index leaf pages.

7.7.3.2 Avoiding Oracle reorganization

Oracle implements PCTFREE only for the data blocks. This value is used in conjunction with *PCTUSED.* PCTUSED sets the minimum percentage of a block that can be used for storing raw data before new rows are added to that block. This means that, after the free space in a data block reaches the PCTFREE value, no new rows are inserted into the block until the percentage of space used falls below PCTUSED.

The default value for PCTFREE is 10 and for PCTUSED it is 40. Both values are related and should be set in conjunction for the different scenarios.

Usually, the less volatile the data is, the lower the PCTFREE parameter can be set; so, data blocks will be completely filled. The higher the PCTUSED value is set, the quicker the page will be reused.

7.7.4 Coping with large, unexpected growth

When the database begins to increase in size more than that which was planned for, some special actions need to be taken.

First, consider partitioning the tables and indexes. This approach can result in some important operational advantages and performance benefits:

- Reduced possibility of data and index corruption
- Balanced I/O
- Easier backup/restore control

Oracle is able to implement this scenario through the creation of *partitioned tables*, that is, tables or indexes are divided into a number of partitions according to the same logical attribute.

Both RDBMSs are able to split ordinary data, indexes, and large objects of one single table into three different tablespaces.

Depending on how large your database is and what kind of workload runs on that machine, just increasing the number of disks, memory, and processors might not be enough in order to survive the growth. Especially for very large DSS systems, you should consider a parallel version of the RDBMS, particularly on an RS/6000 SP machine. DB2 UDB's parallel version is called *DB2 UDB Extended Enterprise Edition* and Oracle's parallel version is named *Oracle Parallel Server*. For more information about parallel databases, refer to Chapter 5, "Parallel databases" on page 81.

7.7.5 Expected growth areas

Both RDBMSs have their own characteristics, concepts, and monitoring methods, but mainly the same growth areas. The following table describes the meaning of each area, its consumption, and its equivalence between both RDBMSs:

Table 8. Equivalence table for expected growth areas

Description	DB2 UDB	ORACLE	Affected area
Records database changes	Log Files	Redo Log Files	Physical
Records system catalog and user data	Data Files	Data Files	Physical
Records index data	Data Files	Data Files	Physical
Used temporary tables' creation and sorts	Temporary Tablespace	Temporary Sort Tablespace	Physical
Stored packages	Package Cache	Library Cache	Logical
Database object's definitions	Catalog Cache	Data dictionary	Logical
Application's allocated resources	Agent Private Memory	Memory Global Area	Logical
Data block copies	Bufferpool	Database Buffer Cache	Logical

Description	DB2 UDB	ORACLE	Affected area
Last/Current data value	Log Buffer	Redo Log Buffer	logical
Last data value	Log Buffer	Rollback Segments	logical

The database growth will really depend on how users and applications are consuming the resources. The monitoring task will indicate which area needs a better tuning.

7.7.6 Loading large amounts of data

Loading large amounts of data can turn into a problem when indexes are defined over a table. Although both RDBMSs have special programs designed to speed up the loading process for large amounts of data, the re-creation of the indexes can still turn into a bottleneck.

Since DB2 UDB V6.1, the Load utility loads data into the table and then creates the index pages for the loaded rows only. This does not influence performance.

Oracle works with a tool called *SQL*Loader*. It works in two different ways: *Conventional path* (use SQL `INSERTS`) and *direct path* (directly writes the external data into database blocks). For both the conventional path and the direct path, SQL*Loader loads data into the table and rebuilds the index for the whole table. When the number of rows being loaded is large compared to the size of the table, this is an acceptable behavior. However, if the number of loaded rows is relatively small, the time required to rebuild the indexes may be excessive. This index rebuild can be avoided through the use of one of the following options:

- Drop the indexes before loading the data
- Mark the indexes as *Index Unusable* before loading the data and use DB2 UDB's SKIP_UNUSABLE_INDEXES option
- Use the DB2 UDB SKIP_INDEX_MAINTENANCE option (only applies to the direct path method)

Another way to avoid full re-indexing is to use Oracle partitioned tables.

7.8 Performance versus availability

The ideal RDBMS scenario is that the database is serving all the connected applications and users in less time than it is expected, twenty-four hours a

day, seven days a week. However, some undesirable and unpredictable factors could cause system downtime, thus, invalidating the ideal scenario. Especially on a machine that holds all the company's important information, the database availability should be one of the main concerns. However, the more security you implement for better availability purposes, the more the overall system performance will usually be impacted.

The administrators should be looking for the perfect balance point between performance and database availability.

For every RDBMS, there are some areas where availability can be improved but with an expected overall performance decrease, for example:

- Disk protection
  The performance will be directly affected by how the disks are set for data storage. Usually the two most often implemented solutions are data mirroring and RAID 5. Please refer to Chapter 8, "Designing a disk subsystem" on page 149 for further information about disks.
- Online backup
  This is the only possible way of taking backup copies while users and applications are connected to the database. It might cause a little overall performance decrease, but it is the only choice for a non-stop system. Please refer to Chapter 2.8.4, "Online and off-line backup" on page 38.
- High Availability Cluster Multiprocessing (HACMP)
  HACMP is an application that can link up to 32 RS/6000 servers or SP nodes in a cluster. Clustering servers enables parallel access to the data and provides redundancy and fault resilience. HACMP also allows administrators to perform hardware, software, and other maintenance activity while the applications continue to run on other machines. The HACMP implementation is highly recommended for 24x7 environments. This chapter will not cover the implementation and use of HACMP.

On the other hand, when performance is increased, availability sometimes suffers. An example is the use of the DB2 UDB load tool, where a performance increase can be achieved, but the database availability can be compromised. The load tool can be used for inserting rows into a table without recording the insert operation in the log files. Although the insert rate significantly increases, a failure in this process can put the tablespace where the table resides in *load pending* state; so, all the tables within that particular tablespace are not accessible. Fortunately, the `TERMINATE` option of the `LOAD` command can roll back the operation and set the involved tables and tablespaces into a normal (available) state.

All these areas can represent a success factor or a performance constraint depending on how they are designed, monitored, and tuned.

Note

Each company will have different performance and availability needs. However, be aware that several availability methods also bring a decrease of performance.

7.9 Production, development, and testing on the same machine

We can classify machines in four different groups according to their functionality:

- Production
- Development
- Testing
- Hybrid

Each one will have an unique and essential role as well as a different system configuration.

7.9.1 Production

This is the machine that holds the company's vital data. The production databases, where all the applications and final users connect to, is usually required to be available 100 percent of the time with an impressive response time.

The production machine must be the most reliable and stable among all the others. Not only must the database and the operating system be monitored, but also the hardware.

Even the worst possible situation that could lead the database to an unexpected stop must have a pre-defined emergency plan already set in order to minimize the downtime. This can be achieved with good and tested backups, as well as with disk protection implementations, such as RAID 5 or mirroring. Also, the High Availability Cluster Multi-Processing (HACMP), an IBM software solution, provides a machine's high-availability through redundancy and shared resource access.

The administrator must focus the performance and tuning efforts onto this machine. The monitoring and performance tasks must be implemented in

order to achieve the best database response time through the efficient use of the operating system and hardware resources. This is the main reason why you should not have any other testing or development load interfering with your production machine.

The production concept implies that this machine holds the fundamental data for the department, or even for the whole company. A severe database downtime on this machine can be the bottleneck for the company's business success. The production machine must have enough, and well consumed, resources as well as administrators committed to the monitoring and performance goals in order to support these characteristics.

Note

In order to reduce the production machine downtime caused by an unexpected problem, it is recommended that the database and system administrators handle this machine with extreme care and always have a tested backup set of the operating system and database available.

7.9.2 Development

This must be a separate small system used exclusively by the development personnel.

Although some basic security recommendations can be taken into account, this is the machine where symptoms, such as running out of disk space and system outages, will occur. This basically happens because the developers are running untested code, and all of the load generated by these tests can directly interfere with CPU, disk, and memory utilization. A lot of times these experiments can easily cause a system havoc, and that is exactly what this machine is about: To show the developers, through the output and effect of their applications, what must be changed in their code and eliminate any possible failure when going into production.

Usually a machine with few CPUs, disk, and memory resources is used as the development machine due to the following facts:

- Very uneven machine resource demand
- Expected instability
- Disconnected from the production environment

7.9.3 Testing

The test system must be a separate, medium sized system where new products, fixes to products, and final user code can be tested before placing these items into the production system. The system can also be used for training purposes.

It should contain a reasonable amount of production data in order to simulate how the code would run with real production mass data. Due to these characteristics, the test machine can be smaller than the production machine but must have enough resources available for running at a reasonable speed.

The test system should never run on the production machine due to the unpredictable behavior of its untried changes.

7.9.4 Hybrid machines

Based on the different characteristics, it is primarily recommended that each scenario is put onto separate machines so that they do not interfere with each other.

However, there are some special situations where the system and database administrators are asked to share the same machine for different scenarios. In this particular case, it is recommend that only development and testing are put together. Once the production machine plays a vital role for the company, this machine should not be exposed to possibly hazardous testing and development failures.

7.10 AIX and RDBMS upgrades

It is always a good practice to have the system on the most up-to-date state. Usually, an *update* is different from an *upgrade* depending whether the product version, release, or modification level is altered. Upgrades alter one of the three product identifiers. In this section, we will reference any change made to the product, no matter if it changed the version, the release, or the modification level, as an upgrade.

Each new upgrade introduces new and useful improvements as well as fixes for occasionally reported problems.

An upgrade could be recommended in one of three cases:

- Instructed by the IBM supporting team in order to fix a known defect
- A new desired feature is available through the upgrade

- Prevent problems by keeping all the products always up-to-date

It is important to be aware that each system has a different reaction when you upgrade the products.

It is always a good and recommended practice to upgrade AIX and RDBMSs following this sequence:

1. Run the AIX utility perfpmr
   (please refer to 14.1, “Perfpmr - the performance data collection tool” on page 345)
2. Back up your database twice
3. Back up the rest of the system twice
4. Choose one of the products for upgrading (do not upgrade multiple products at the same time)
5. Upgrade the product
6. Test the upgrade with your own pre-defined *proof of concept* test programs
7. Back up the rest of the system twice again
8. Run the AIX utility perfpmr again

It is also highly recommended to upgrade the testing environment first and, only when the *proof of concept* programs indicate that the environment is stable, upgrade the production environment. However, on systems where there is no other machine to test the upgrade but the production system, extra care should be taken, especially with the integrity of the backup image. For this kind of machine configuration, it is suggested that a High Availability Cluster Multi-Processing (HACMP) is implemented. HACMP is an IBM software solution that provides a machine’s high availability through redundancy and shared resource access.

Usually, the AIX operating system is upgraded by the system administrator, while the RDBMSs are upgraded by the database administrator.

In order to upgrade AIX and DB2 UDB, please contact your local IBM Support Team. DB2 UDB upgrades can also be downloaded from the following Web-site:
`ftp://ftp.software.ibm.com/ps/products/db2/fixes/english-us/db2aixv61/`

In order to request upgrades for Oracle RDBMS, please contact your local Oracle Support Team.

Chapter 8. Designing a disk subsystem

When designing a system for performance, many different factors must be taken into consideration. You will often find that changing one facet of the design can have unexpected, sometimes undesirable, effects on the overall performance of the system. This is especially true when deciding on how to lay out the disk subsystem for optimal performance. Some of the factors that we will explore in this chapter include:

- Bandwidth of disk adapters and system bus
- Physical database layout
- Configuration of the disk hardware
- Workload characteristics of the application
- Operating system settings including Logical Volume Manager (LVM) considerations
- Performance implications of different disk technologies (SSA, striping, mirroring, RAID and normal disks)

A more detailed discussion on implementing a disk storage system for systems running Oracle databases can be found in the IBM white paper *Configuring and Tuning IBM Storage Systems in an Oracle Environment*. IBM employees can obtain the white paper on the IBM Intranet at the following URL:

`http://w3.developer.ibm.com/depts/spra/ORACLE/downloads/Oracle_Stor.PDF`

The white paper can also be obtained from the IBM/Oracle International Competency Center by sending an E-mail request to: `ibmoracl@us.ibm.com`

8.1 Disk subsystem design approach

For many systems, the overall performance of an application is bound by the speed at which data can be accessed from disk. Designing and configuring a disk storage subsystem for performance is a complex task that must be carefully thought out before the first disk is purchased. Some of the factors that must be considered include:

- Performance versus availability
  A decision must be made early on as to which is more important: I/O performance of the application or application integrity and availability. Increased data availability often comes at the cost of decreased system performance and vice versa.

- Application workload type
  The I/O workload characteristics of the application should be fairly well understood prior to implementing the disk subsystem. Different workload types most often require differently configured disk subsystems in order to provide acceptable I/O performance. Further descriptions of database workload types can be found in Chapter 3, “Types of Workload” on page 31.
- Required disk subsystem throughput
  The I/O performance requirements of the application should be defined up front, as they will play a large part in dictating both the physical and logical configuration of the disk subsystem. In database environments, this is typically expressed in transactions per second or minute (tps/tpm).
- Required disk space
  Prior to designing the disk subsystem, the disk space requirements of the application should be well understood. Guidelines for estimating the amount of disk space that will be required can be found in 6.6, “Disk goals and sizing” on page 112.
- Cost
  While not a performance related concern, overall cost of the disk subsystem most often plays a large part in dictating the design of the system. Just as with performance and availability, cost and performance are inversely related: You must sacrifice one in order to achieve gains in the other.

8.2 Bandwidth related performance considerations

The bandwidth of a communication link, such as a disk adapter or bus, determines the maximum speed at which data can be transmitted over the link. When describing the capabilities of a particular disk subsystem component, performance numbers are typically expressed in maximum or peak throughput, which often do not realistically describe the true performance that will be realized in a real world setting. In addition, each component will most likely have different bandwidths, which can create bottlenecks in the overall design of the system.

The bandwidth of each of the following components must be taken into consideration when designing the disk subsystem:

- Disk devices
  The latest SCSI and SSA disk drives have maximum sustained data transfer rates of 14-20 MB per second. Again, the real world expected rate will most likely be lower depending on the data location and the I/O workload characteristics of the application. Applications that perform a

large amount of sequential disk reads or writes will be able to achieve higher data transfer rates than those that perform primarily random I/O operations.

- Disk adapters
  The disk adapter can become a bottleneck depending on the number of disk devices that are attached and their use. While the SCSI-2 specification allows for a maximum data transfer rate of 20 MB/sec, adapters based on the UltraSCSI specification are capable of providing bandwidth of up to 40 MB/sec. The SCSI bus used for data transfer is an arbitrated bus. In other words, only one initiator or device can be sending data at any one time. This means the theoretical maximum transfer rate is unlikely to be sustained. By comparison, the IBM SSA adapters use a non-arbitrated loop protocol, which also supports multiple concurrent peer-to-peer data transfers on the loop. The current SSA adapters are capable of supporting maximum theoretical data transfer rates of 160 MB/sec.
- System bus
  The system bus architecture used can further limit the overall bandwidth of the disk subsystem. Just as the bandwidth of the disk devices is limited by the bandwidth of the disk adapter to which they are attached, the speed of the disk adapter is limited by the bandwidth of the system bus. The industry standard PCI bus is limited to a theoretical maximum of either 132 MB/sec (32-bit) or 528 MB/sec (64-bit).

8.3 Physical database layout considerations

Deciding on the physical layout of the database is one of the most important decisions to be made when designing a system for optimal performance. The physical location of the database's datafiles is critical to ensuring that no single disk, or group of disks, becomes a bottleneck in the I/O performance of the application. In order to minimize their impact on disk performance, heavily accessed tables and their corresponding datafiles should be placed on separate disks, ideally under different disk adapters.

There are several ways to ensure even data distribution among disks and adapters, including operating system level data striping, hardware data striping (RAID), and manually distributing the database data files among the available disks. This section is concerned with the manual method of distributing the data. Operating system level data striping techniques are covered in Section 8.4.1, "Physical Partition striping versus LVM fine striping" on page 154, while hardware level data striping using RAID is addressed in

Section 8.6, “RAID Levels overview and performance considerations” on page 161.

8.3.1 Database datafile distribution

Manual distribution of the database datafiles is one the most widely used methods for attempting to evenly distribute the I/O workload of an application. This can be attributed to the fact that, unlike OS level and hardware data striping techniques, manually distributing the database datafiles across multiple disks does not require any additional hardware or operating system capabilities.

One major drawback to this approach is that as the database data access patterns become skewed due to data growth, more and more manual effort is required to manually distribute the I/O among the disks. In contrast to OS level and hardware data striping techniques, manual datafile distribution often requires significant downtime in order to move tables and indexes to different disks, as the database (or at least the affected tables and indexes) must be made unavailable while the work is being performed.

When using the manual method for distributing data, the following I/O intensive database datafiles should be placed on separate disks for optimal performance:

- Data dictionary tablespace - SYSTEM (ORACLE) or SYSCATSPACE (DB2 UDB) used to maintain internal database operation information, such as database structure and performance statistics.
- Temporary tablespace used for performing sort operations that cannot be done in memory.
- Rollback segments tablespace holds the prior value of the data in order to guarantee read consistency.
- Redo log files record every change made to the database. This has heavy sequential I/O activity.
- Database datafiles with heavily accessed tables.
- Database datafiles with heavily accessed indexes.

The database data tables and indexes should be placed on separate disks attached to separate disk adapters in order to avoid I/O contention, especially for those tables that are frequently joined (`JOIN`) in SQL queries. The database redo logs are the most I/O intensive datafiles in the entire system due to the fact that they record every change made to the database. Due to their importance in ensuring the integrity of the system, they are often mirrored via the RDBMS software. Since the updates to the redo logs are

performed sequentially, both the redo logs and their mirrors should be placed on dedicated disks, each attached to a separate disk adapter. This will ensure that there is no other disk activity taking place that would interfere with their update.

8.4 Logical Volume Manager (LVM) Concepts

Many modern UNIX operating systems implement the concept of a Logical Volume Manager (LVM) that can be used to logically manage the distribution of data on physical disk devices. The AIX LVM is a set of operating system commands, library subroutines, and other tools used to control physical disk resources by providing a simplified logical view of the available storage space. Unlike some competitor's LVM offerings, the AIX LVM is an integral part of the base AIX operating system provided at no additional cost.

Within the LVM, each disk or physical volume (PV) belongs to a *volume group* (VG). A volume group is a collection of 1 to 32 physical volumes (1 to 128 in the case of a big volume group), which can vary in capacity and performance. A physical volume can belong to only one volume group at a time. A maximum of 255 volume groups can be defined per system.

When a volume group is created, the physical volumes within the volume group are partitioned into contiguous, equal-sized units of disk space known as *physical partitions* (PP). Physical partitions are the smallest unit of allocatable storage space in a volume group. The physical partition size is determined at volume group creation, and all physical volumes that are placed in the volume group inherit this size. The physical partition size can range from 1 to 1024 MB but must be a power of 2. If not specified, the default physical partition size in AIX 4.3 is 4 MB for disks up to 4 GB but must be larger for disks greater than 4 GB due to the fact that the LVM, by default, will only track up to 1016 physical partitions per disk (unless you use the `-t` option with `mkvg`, which, however, reduces the maximum number of physical volumes in the volume group). Table 9 lists typical physical partition sizes for 2.2, 4.5, 9.1, 18.2, and 36.4 GB physical disks.

Table 9. Typical physical partition sizes for varying physical disk sizes

Physical disk size	Physical partition size
2.2 GB	4 MB
4.5 GB	8 MB
9.1 GB	16 MB
18.2 GB	32 MB

Physical disk size	Physical partition size
36.4 GB	64 MB

After adding a physical disk to a volume group, in order to use the storage space you must create *logical volumes* (LV). Logical volumes define disk space allocation at the physical partition level. They can reside on many different non-contiguous physical partitions, thereby, allowing them to span physical disks. At the operating system level, logical volumes appear to applications as one single, contiguous disk.

When creating logical volumes, you must specify the number of *logical partitions* to allocate. Each logical partition maps to one, two, or three physical partitions depending on how many copies of the data you want to maintain. This allows for mirroring of the data either on the same physical disk or different disks in the same volume group.

8.4.1 Physical Partition striping versus LVM fine striping

Most disk I/O performance bottlenecks in a database environment are the result of a few *hot* tables or datafiles receiving a disproportionate amount of the I/O activity in the system. While manually distributing the database datafiles among physical disk devices can alleviate some of these bottlenecks, this is typically not a good solution for large, heavily accessed databases for the reasons documented in Section 8.3.1, “Database datafile distribution” on page 152.

For databases that generate a large amount of I/O activity, no amount of system memory, database buffers, or external cache can shield the application from performance problems associated with skewed data access patterns. For these types of databases, the most viable solution may be to consider distributing the I/O load across a number of physical disk drives through the use of *data striping* techniques. Since the data is located on more than one physical device, data striping allows a single disk I/O request to be divided into multiple parallel I/O operations.

The AIX LVM provides two different techniques for striping data: *Physical Partition (PP) striping* and *LVM striping*. In Figure 17 on page 118, the numbers represent the sequence of data blocks for a given file using both PP and LVM striping.

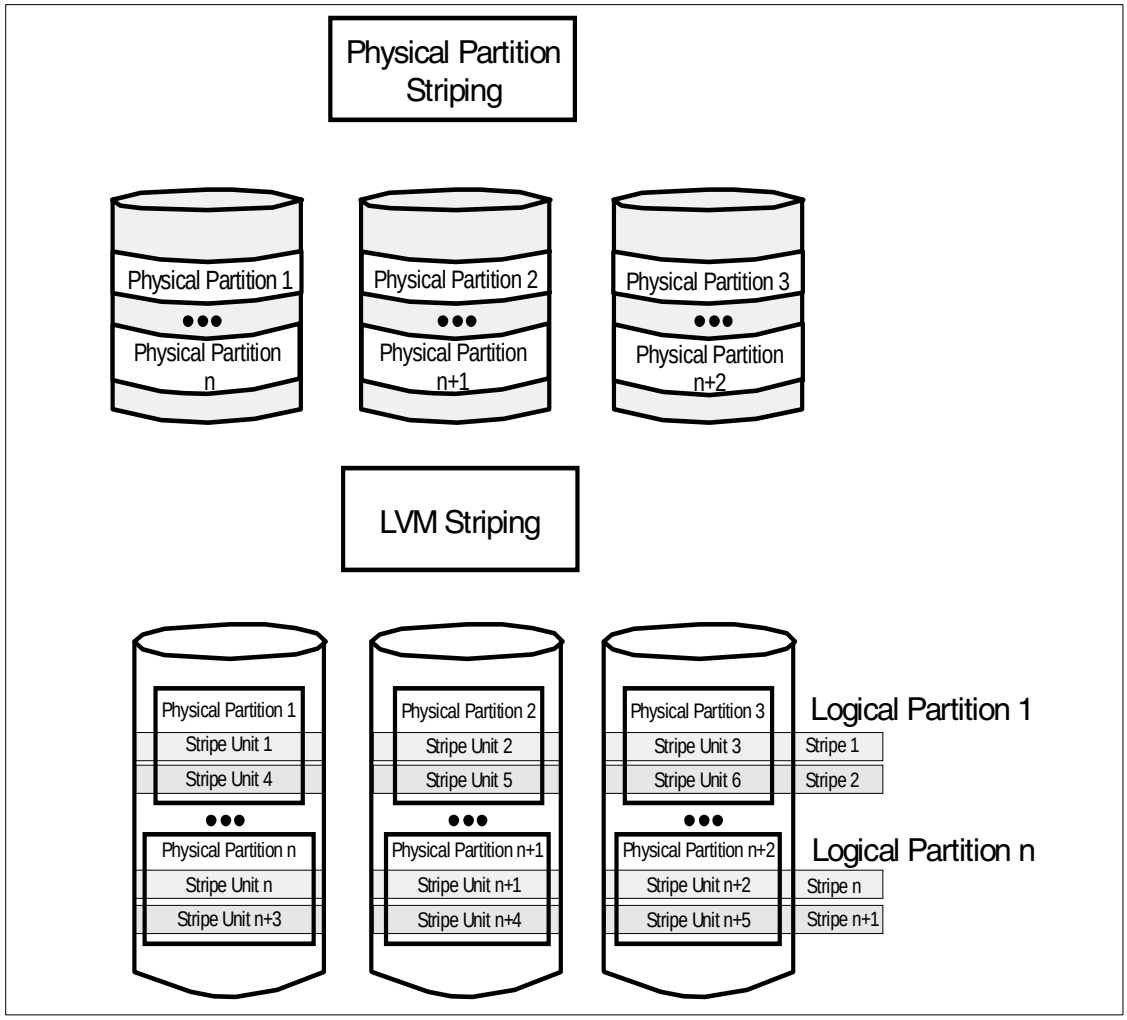


Figure 26. Physical Partition and LVM striping example

Physical Partition striping refers to the technique of spreading the physical partitions of a logical volume across two or more physical disk drives. With PP striping, the size of the data stripe is the size of the physical partition, which is typically 4, 8, or 16 MB in size. This technique works well in environments that are characterized by a large amount of primarily random I/O operations, such as OLTP applications.

LVM striping, also known as *fine striping*, likewise attempts to distribute the I/O load by placing data stripes on multiple physical disks. However, LVM

striping differs from PP striping in its use of a more granular or fine data stripe. With LVM striping, each logical partition of a logical volume is broken up into multiple stripe *units* and distributed among all of the physical devices that contain part of the logical volume. The stripe unit size must be a power of two in the range 4 KB to 128 KB and is specified when the logical volume is created.

LVM striping works best in environments that perform many sequential read and write operations against large datafiles due to the performance benefits of *sequential read ahead*. Sequential read ahead occurs when either the RDBMS or the AIX Virtual Memory Manager (VMM) detects that the file is being accessed sequentially. In this case, additional disk reads are scheduled against the file in order to pre-fetch data into memory. This makes the data available to the program much faster than if it had to explicitly request the data as part of another I/O operation. Sequential read ahead is only available for files residing on JFS file systems and has no meaning for raw devices (raw logical volumes). DSS and batch workloads are good candidates for LVM striping.

Note

Prior to AIX version 4.3.3, logical volumes could not be mirrored and striped at the same time. Logical volume mirroring and striping combines the data availability of RAID 1 with the performance of RAID 0 entirely through software. Volume groups that contain striped and mirrored logical volumes cannot be imported into AIX Versions 4.3.2 and below.

8.4.2 Use of LVM policies

The AIX LVM provides a number of facilities or *policies* for managing both the performance and availability characteristics of logical volumes. The policies that have the greatest impact on performance are: *Intra-disk allocation*, *inter-disk allocation*, *write scheduling,* and *write-verify* policies.

8.4.2.1 Intra-disk allocation policy

The intra-disk allocation policy determines the actual physical location of the physical partitions on disk. The disk is logically divided into the following five concentric areas: Outer edge, outer middle, center, inner middle, and inner edge.

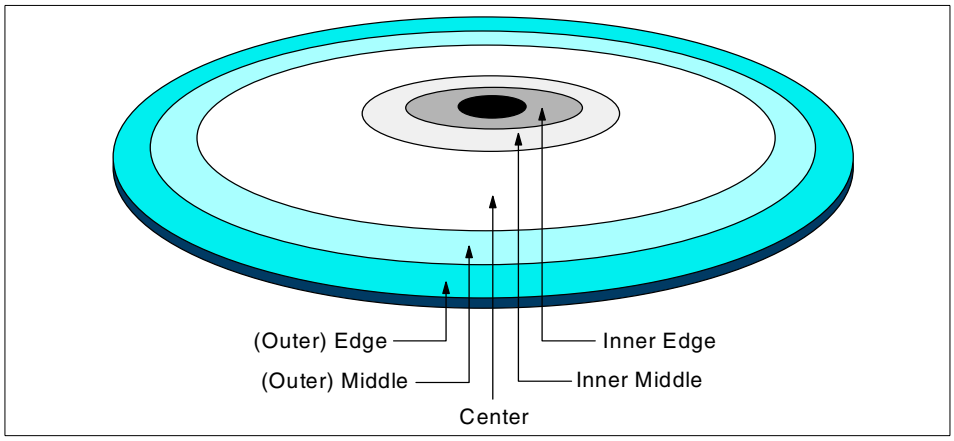


Figure 27. Physical Partition mapping

Due to the physical movement of the disk actuator, the outer and inner edges typically have the largest average seek times and are a poor choice for application data that is frequently accessed. The center region provides the fastest average seek times and is the best choice for applications that generate a significant amount of I/O activity. The outer and inner middle regions provide better average seek times than the outer and inner edges but worse seek times than the center region.

As a general rule, when designing a logical volume strategy for performance, the most performance critical data should be placed as close to the center of the disk as possible. There are, however, two notable exceptions:

1. Applications that perform a large amount of sequential reads or writes experience higher throughput when the data is located on the outer edge of the disk due to the fact that there are more data blocks per track on the outer edge of the disk than the other disk regions.
2. Logical volumes with Mirrored Write Consistency (MWC) enabled should also be located at the outer edge of the disk, as this is where the MWC cache record is located. Please refer to Section 8.7, “Use of Mirror Write Consistency (MWC)” on page 167 for further information concerning the effect of MWC on logical volume performance.

8.4.2.2 Inter-disk allocation policy

The inter-disk allocation policy is used to specify the number of disks that contain the physical partitions of a logical volume. The physical partitions for a given logical volume can reside on one or several disks in the same volume group depending on the setting of the *Range* option:

- The *maximum* range setting attempts to spread the physical partitions of a logical volume across as many physical volumes as possible in order to decrease the average access time for the logical volume.
- The *minimum* range setting attempts to place all of the physical partitions of a logical volume on the same physical disk. If this cannot be done, it will attempt to place the physical partitions on as few disks as possible. The minimum setting is used for increased availability only and should not be used for frequently accessed logical volumes. If a non-mirrored logical volume is spread across more than one drive, the loss of any of the physical drives will result in data loss. In other words, a non-mirrored logical volume spread across two drives will be twice as likely to experience a loss of data as one that resides on only one drive.

The physical partitions of a given logical volume can be mirrored to increase data availability. The location of the physical partition copies is determined by the setting of the *Strict* option. When *Strict = y*, each physical partition copy is placed on a different physical volume. When *Strict = n*, the copies can be on the same physical volume or different volumes. When using striped and mirrored logical volumes in AIX 4.3.3 and above, there is an additional partition allocation policy known as *Super Strict*. When *Strict = s*, partitions of one mirror cannot share the same disk as partitions from a second or third mirror, thus, further reducing the possibility of data loss due to a single disk failure.

In order to determine the data placement strategy for a mirrored logical volume, the settings for both the Range and Strict options must be carefully considered. As an example, consider a mirrored logical volume with range setting of *minimum* and a strict setting of *yes*. The LVM would attempt to place all of the physical partitions associated with the primary copy on one physical disk, with the mirrors residing on either one or two additional disks, depending on the number of copies of the logical volume (2 or 3). If the strict setting were changed to *no*, all of the physical partitions corresponding to both the primary and mirrors would be located on the same physical disk.

8.4.2.3 Write-scheduling policy

When mirrored copies of the data are maintained by the LVM, the setting of the mirrored write-scheduling policy determines the sequence of the write operations to the logical volume. Mirrored writes can be either *parallel* or *sequential*.

The sequential write-scheduling policy writes the physical partitions for a mirrored logical volume in the sequence *primary*, *secondary,* and *tertiary,* where primary represents the first copy of the logical volume, secondary the

second, and tertiary the third. A write request for a given copy must complete prior to updating the next copy in the sequence. Read requests are first directed to the primary copy. If that copy cannot be accessed due to a drive failure or physical corruption, the request is redirected to the secondary copy and so forth. While the redirected read request is being processed, the LVM automatically attempts to correct the copies on which the read failed through a process known as *bad block relocation*.

The parallel write-scheduling policy schedules the write operation to each of the copies at the same time. The write request is satisfied when the copy that takes the longest to update is finished. The parallel write-scheduling option provides the best performance, as the duration of the write request is limited only by the speed of the slowest disk and not the number of copies that must be updated. Read requests are directed to the copy that can be accessed in the shortest amount of time, thereby realizing the same performance gains as write requests. Just as with the sequential-write policy, failed read requests will automatically initiate bad block relocation.

8.4.2.4 Write-verify policy

When the write-verify policy is enabled, all write operations are validated by immediately performing a follow-up read operation of the previously written data. An error message will be returned if the read operation is not successful. The use of write-verify enhances the integrity of the data but can drastically degrade the performance of disk writes.

8.4.2.5 Recommendations for performance optimization

As with any other area of system design, when deciding on the LVM policies to be used, a decision must be made as to which is more important: Performance or availability. The following LVM policy guidelines should be followed when designing a disk subsystem for performance:

- When using LVM mirroring:
  - Use a parallel write-scheduling policy.
  - Allocate each logical partition copy on a separate physical disk by using the Strict option of the inter-disk allocation policy.
- Disable write-verify
- Allocate heavily accessed logical volumes near the center of the disk, with the exceptions noted in Section 8.4.2.1, “Intra-disk allocation policy” on page 156.
- Use a intra-disk allocation policy of *maximum* in order to spread the physical partitions of the logical volume across as many physical disks as possible.

8.5 Raw logical volumes versus Journaled File Systems (JFS)

There has been a long standing debate surrounding the use of raw logical volumes (raw devices) versus Journaled File Systems (JFS), especially in database environments. Advocates of raw logical volumes stress the performance gains that can be realized through their use, while JFS supporters emphasize the ease of use and manageability features of file systems. As with many other aspects of system design, a decision must be made as to which is more important: Performance or manageability.

In order to better understand the performance advantages associated with raw logical volumes, it is helpful to have an understanding of the impact of the JFS file system cache. Most UNIX file systems set aside an area of memory to hold recently accessed file data, thereby, allowing a physical I/O request to be satisfied from memory instead of from disk. In AIX, this area of memory is known as the *buffer cache*. If an application requests data that is not already in memory, AIX will read the data from disk into the buffer cache and then copy the data to a user buffer so that it can be used by the application. Therefore, each read request translates into a disk read followed by a copy of the data from the buffer cache to the user buffer.

Because the data is read from memory, I/O requests can be satisfied in nanoseconds instead of the milliseconds that would be required in order to fetch the data from disk. In addition, AIX JFS file systems employ the use of a sequential read-ahead mechanism to pre-fetch data into memory when it is determined that a file is being accessed sequentially.

In non-database environments, the AIX buffer cache can significantly reduce I/O wait time for heavily accessed files. However, the performance benefits of file system caching in database environments are not so clear. This is due to the fact that most modern RDBMS systems also allocate a region of memory for caching frequently accessed data. The end result when using JFS file systems is that the data is double-buffered: Once in the file system buffer cache and once in the RDBMS cache. In most cases, the extra memory used by the file system buffer cache could be better utilized by the database buffers.

The primary benefit of raw logical volumes is that they bypass the AIX file system buffer cache entirely by directly accessing the underlying logical device. The extra memory saved by eliminating the file system cache can then be allocated to the database to increase the data buffers. In addition, overall CPU utilization is decreased due to the fact that the system no longer has to copy the data from the file system cache to the user buffers. Another benefit of raw logical volumes is that there is no inode management

overhead, as opposed to JFS where the inode is locked when the file is accessed.

The main drawback of using raw logical volumes lies in the increased administration costs associated with their use. Since raw logical volumes do not exist as files at the UNIX level, many of the traditional tools and utilities for managing data will not work. Backup and recovery operations can be especially difficult when using raw logical volumes. Many third party vendor backup applications (such as the IBM ADSM) cannot directly read raw logical volumes and must rely on the UNIX `dd` command to copy the raw data to a UNIX filesystem prior to backing up the data. Restores are equally complicated as the data must first be restored to a UNIX filesystem and then copied to the raw logical volume. If this approach is used, additional disk space will be required for the JFS filesystems used to temporarily hold the data contained in the raw logical volume. However, if the raw logical volumes can be backed up directly to a locally attached tape drive using the `dd` command, this will not be an issue.

Many raw logical volume benchmarks point to an overall disk I/O throughput gain of 5-30 percent when compared to JFS file systems. However, the actual performance gains that can be realized in a typical database environment will vary depending on the I/O workload mix of the application. Applications that perform a large amount of random I/O operations, such as OLTP systems, benefit the most from the use of raw logical volumes. Applications that perform a large amount of sequential I/O operations, such as DSS systems, benefit from the sequential read ahead feature of JFS file systems.

8.6 RAID Levels overview and performance considerations

Redundant Array of Independent Disks (RAID) is a term used to describe the technique of improving data availability through the use of *arrays* of disks and various data striping methodologies. A disk array is a group of physical disk drives used simultaneously to achieve higher data transfer and I/O rates than those available through the use of one single drive. IBM was responsible for much of the initial research and development into the use of RAID, with the first patent being issued in 1978.

The initial focus of RAID research was to improve performance while at the same time reducing the overall cost per unit of storage. Further research emphasized the improved data reliability and fault tolerance that characterizes modern RAID systems.

The alternative to RAID disks is a set of disks connected to the system in which logical volumes are placed, and any one logical volume is entirely on one disk. This is often called JBOD, meaning Just a Bunch of Disks.

Within the RAID architecture, there are varying degrees of data reliability and performance, known as RAID *Levels*. Depending on the RAID Level, data can be either mirrored or striped. Data redundancy is provided through data mirroring, which maintains two copies of the data on separate physical disks. Data striping involves distributing the data among several disks by splitting it into multiple sequential data blocks and writing them to each of the drives in the array in parallel. In addition, most of the RAID Levels create parity information that can be used to reconstruct data on a particular drive in the event of a failure. The standard RAID specification provides for Levels 0-6, although some vendor specific implementations exist, such as EMC's RAID-S.

Note

The following overview contains performance-related comparisons of the different RAID Levels. These comparisons, and the resulting recommendations, are meant to compare and contrast the performance characteristics of different RAID Levels only and are not meant to serve as a comparison of RAID versus LVM performance.

8.6.1 RAID Level 0

RAID 0, referred to as data striping, differs from the other RAID implementations in that it does not offer any form of data redundancy. RAID 0 splits data into chunks and then writes or *stripes* the data sequentially across all of the disks in the array. This implementation offers the following performance advantages:

- Parallel I/O streams to multiple drives allow for higher data transfer rates for sequential read/write operations
- Increased throughput of random disk accesses due to the distribution of data onto multiple disks

The primary disadvantage of a RAID 0 configuration is that, should a single disk in the array fail, all of the data in the array will become unusable due to the fact the data cannot be reconstructed from the remaining drives. RAID 0 should, therefore, be used for applications that require a high level of performance but which do not have very stringent data availability requirements.

8.6.2 RAID Level 1

RAID 1, also known as disk mirroring, uses data mirroring to achieve a high level of redundancy. In a RAID 1 configuration, two copies of the data are kept on separate disks, each mirroring the other. In the event of a single disk failure, all read/write operations will be redirected to the mirrored copy of the data. RAID 1 configurations are the most expensive of any of the other solutions due to the fact that twice as many disks are required in order to mirror the data.

Read performance of a RAID 1 configuration implemented via the AIX LVM is enhanced due to the fact that, should the primary copy be busy, read requests are directed to the mirror copy. Write performance can be slower than in non-RAID implementations depending on the write scheduling policy selected through the LVM: *Parallel* or *sequential*.

Using the parallel scheduling policy, the writes to all copies of the data are initiated at the same time or in *parallel*. The write operation is considered complete when the copy that takes the longest time to update is finished. This is the fastest but less reliable option, as a failure to write to one of the copies may go undetected for a period of time.

Under the sequential scheduling policy, the update of the mirror is not started until the write to the primary copy has successfully completed. This is the more reliable, but slower, of the two methods.

In addition, the use of mirror write consistency can have an impact on the performance since potentially four disk write operations are performed for each LVM write operation. See Section 8.7, “Use of Mirror Write Consistency (MWC)” on page 167 for more details.

8.6.3 RAID Level 2 and Level 3

RAID Level 2 and Level 3 both break data into multiple chunks or segments and evenly distribute it across several physical disks. Striping in RAID 2 and RAID 3 occurs at the bit or multi-byte level. During a read operation, multiple simultaneous requests are sent to each disk causing all of the disk actuators - the arm that holds the read/write head for the disk - to move in parallel. This limits the number of concurrent I/O operations in the array to one.

In order to provide data redundancy, RAID 2 and 3 configurations require parity information to be written for each write operation performed. While RAID 2 can distribute the parity information across multiple drives through the use of an encoding technique known as the Hamming method, RAID 3 uses only one drive for the parity. If one drive in the array fails, the data can still be

accessed and updated using the parity information. However, the system will operate in a degraded mode until the drive is fixed due to the time required to dynamically reconstruct the data located on the failed drive using the parity information.

Note

The AIX LVM does not directly support RAID Level 2 or RAID Level 3.

8.6.4 RAID Level 4

RAID 4 is very similar to RAID 3 in that it stripes the data across multiple physical disks in the array. The primary difference is that the striping increment is a block or record instead of the bit or byte method used by RAID 3 configurations. By virtue of the larger data increment used to create the stripe, reads can be matched to the one physical disk that contains the requested data. This allows both simultaneous and independent reads to be processed.

As in RAID 3, a single parity disk is used for data redundancy. This can create a performance bottleneck for write operations, as requests to update the sole parity disk cannot be processed in parallel. Due to the performance problems associated with the single parity disk and RAID 4's similarity to RAID 5, RAID 4 is not a commonly used or recommended configuration.

Note

The AIX LVM does not directly support RAID Level 4.

8.6.5 RAID Level 5

Instead of having a dedicated parity disk, RAID 5 interleaves both data and parity on all disks. In a 4+P RAID 5 array, five disks are used for data and parity combined. Four-fifths of the space on those disks is used for data and one-fifth of the space is used for parity. In RAID 5, the disks can be accessed independently of one another, and it is possible to use a large stripe size; so, most data transfers involve only one data disk. This enables multiple concurrent accesses, thereby, giving higher throughput for OLTP or other random workloads.

Due to the way in which parity data is typically generated for RAID 5, there is a write penalty associated with write access. Random write I/Os usually result in four actual I/O operations:

1. Read the old data
2. Read the old parity
3. Write the new data
4. Write the new parity

Some IBM RAID 5 implementations, such as the SSA RAID adapters, incorporate full or partial stripe write algorithms for sequential writes. This reduces the number of I/O operations required. Also, the use of read/write cache in the adapter can mask the write penalty for many workloads either by getting a cache hit during the data read operation or by caching the writes. It is important to note that there is some form of write penalty associated with any redundant RAID architecture, including RAID 1. This is due to the fact that some amount of redundant information must be written in addition to the base data.

The IBM PCI SSA RAID adapters can be configured with an optional fast write cache, which dramatically reduces the impact of the write penalty associated with RAID 5 implementations.

8.6.6 RAID 0+1

RAID 0+1, also known as IBM RAID-1 Enhanced or RAID 10, is a combination of RAID 0 (data striping) and RAID 1 (data mirroring). RAID 0+1 provides the performance advantages of RAID 0 while maintaining the data availability of RAID 1. In RAID 0+1 configurations, both the data and its mirror are striped across all the disks in the array. The first stripe is the data stripe, and the second stripe is the mirror, with the mirror being placed on a different physical drive than the data. RAID 0+1 implementations provide excellent write performance, as they do not have to calculate or write parity data. RAID 0+1 can be implemented solely in software (AIX), solely in hardware, or in a combination of hardware and software. Which is the appropriate solution for each implementation depends on overall requirements, such as high availability.

8.6.7 Comparison of RAID Levels

The following chart summarizes the performance and availability characteristics of the different RAID Levels:

Table 10. Comparison of RAID Levels

RAID Level	Capacity	Data Protection	Performance			Cost
			Sequential	Random Read	Random Write	
RAID 0	Very High	None	High	High	High	Low
RAID 1	Moderate	Very Good	Medium-High	Medium-High	Medium	High (can be twice RAID 0)
RAID 3	High	Good	High	Low-Medium	Low-Medium	Medium
RAID 5	High	Good	High	High	Medium	Medium
RAID 0+1	High	Very Good	High	High	High	High

8.6.8 RAID 5 versus AIX LVM mirroring

When deciding on a data protection strategy, most customers narrow their choices down to the two most widely implemented solutions: SSA RAID 5 or LVM mirroring. Both solutions provide a highly robust and reliable data protection mechanism with varying degrees of performance and cost.

When evaluating the performance of a RAID 5 configuration, two important factors should be considered: The number of disks in the array and the read/write ratio of the application that will be using the array. In RAID 5 configurations, transaction performance (especially for reads) is directly related to the number of disks used in the array. As the number of disks in the array increases, so do the number of I/O operations processed/second, up to the limits of the RAID adapter. This is due to the fact that read operations can be processed in parallel across the disks in the array

Note

The number of I/O operations processed/second will decrease if the size of the logical volume that is striped across the array increases. This is due to the fact that the each disk in the array will be required to seek over a larger area of the disk in order to read or write data.

The read/write ratio of the application is the other factor that should be considered when assessing the performance of a RAID 5 configuration. The write penalty associated with RAID 5 configurations that do not utilize a fast write cache can result in severe performance degradation for applications that are write intensive. If the application is characterized by a large number of read operations and relatively few writes, RAID 5 solutions can provide roughly equivalent performance to their mirrored counterparts provided they use sufficiently large disk arrays.

For applications that are not particularly I/O intensive, RAID 5, not regarding the usage of the fast write cache, can provide reasonable performance at a significant cost savings when compared to mirrored solutions. As an example, in a RAID 5 environment, eight disks would be required for seven disks worth of storage: seven for the data and one for the distributed parity. By comparison, a mirrored environment would require 14 disks: Seven for the data and seven for mirrors.

In summary, since both RAID 5 and LVM mirroring solutions provide equally reliable data protection mechanisms, one should focus on the following factors when attempting to choose between the two solutions:

- Performance characteristics of the application - Does the application process many random I/O operations, or does it perform primarily sequential reads and writes?
- I/O response times required for the application - How critical is I/O performance to the operation of the application?
- Cost - Which solution provides the best performance at the lowest cost?

8.7 Use of Mirror Write Consistency (MWC)

Mirror Write Consistency (MWC) is a technique used by the LVM to ensure data consistency between mirrored logical volume copies when a volume group is not varied off-line cleanly. If all LV copies are not kept in sync, a read request will return two different versions of the data depending on which copy was chosen by the LVM.

Note

Mirror Write Consistency is enabled, by default, for mirrored logical volumes. To disable MWC, you must use the `-w n` option of the `mklv` command when creating the logical volume. To disable MWC on previously created logical volumes, use the `-w n` option of the `chlv` command. Please refer to the *AIX Commands Reference*, SBOF-1877, for your particular operating system version for a full description of the `mklv` and `chlv` commands

A Mirror Write Consistency (MWC) log (sometimes referred to as the Mirror Write Consistency cache or MWCC) identifies those Logical Track Group copies that may be inconsistent when a volume group is not varied off-line cleanly. A Logical Track Group (LTG) is a group of 32 4 K blocks (128 K bytes total) within a logical volume (LV). When a volume group is varied back on-line, LVM uses the MWC log to make LTG copies consistent again in all mirrored logical volumes for which MWC is enabled.

An MWC log for each active volume group is maintained in kernel memory. There is a copy of the log in a single 512-byte disk block near the outer edge (prior to the first physical partition) of every physical volume in the volume group. Since the MWC log is written to different disks at different times, the disk copies are not usually identical.

An MWC log has 62 entries. Each entry contains a logical volume minor number and the index of a Logical Track Group within the logical volume. When a write request is received for a mirrored logical volume with MWC enabled, the MWC log in kernel memory is checked for the LTG that will be affected by the write. If an entry for the LTG is found in the MWC log in memory, then the corresponding write requests are allowed to proceed. If there is no entry for the LTG, one is added and the updated MWC log is written to the physical volumes on which the affected LTG copies reside. Meanwhile, the LTG write request is held within the scheduling layer. When the MWC log write requests are complete, the LTG write request is released and allowed to proceed.

Entries are never removed from the MWC log in memory. Instead, a count of the number of writes active to the specified LTG is kept for each entry. When the count is zero, the entry is eligible for reuse. When a new write request arrives for an LTG that is not yet in the log, it replaces the least recently used entry that has a write count of zero. If all 62 entries have non-zero write counts, the LTG write request is queued until the write count of some entry goes to zero.

AIX does not return *iodone* to an application until writes have completed to all mirror copies (or until one or more copies have been marked stale). To synchronize a logical volume after a crash, an RDBMS must reissue all write requests for which iodone had not been received at the time of the crash. Mirrored logical volumes containing JFS logs or file systems must be synchronized after a crash either by forcing a sync (see below) or by enabling MWC prior to the crash.

When a logical volume closes, all MWC log entries referring to the logical volume are cleared, and the MWC log is written to every disk in the volume group.

When a volume group is varied back online, AIX reads the MWC log record from every disk in the volume group and processes each MWC entry in each log. For each unique MWC entry (which contains a logical volume minor number and the index of a Logical Track Group within the logical volume), AIX reads the LTG from the logical volume and writes it back. Duplicate MWC entries that have already been processed are ignored.

The LTG copy that is read depends upon the logical volume's scheduling policy. If the scheduling policy is parallel, AIX directs the read request to the available copy that is least busy. If neither copy is busy, the first is used. If the scheduling policy is sequential, AIX tries each copy in turn, starting with the first, until it finds one that is available.

Since the LTG copy read and write does not necessarily contain the latest data, there is no guarantee that the latest data will be propagated. An RDBMS must, therefore, determine the validity of data in question after a crash (the data for which a write had been issued but for which no iodone had been received at the time of the crash) even if MWC is used to guarantee consistency.

In other words, enabling MWC insures the consistency of logical volume copies but does not protect the integrity of logical volume data. Even when a logical volume is mirrored to improve availability, an RDBMS must still take steps to provide for data integrity.

If MWC must be used, and write throughput is important (to optimize RDBMS update performance, for example), it is unwise to implement volume groups with anything close to the maximum number of physical volumes (128) currently allowed. When every logical volume in a volume group is double-mirrored with MWC enabled, write I/O can be in progress to, at most, 62 LTGs at any given instant. Since each LTG has two copies, write I/O can be in progress to, at most, 124 physical LTG copies (at most,124 physical

volumes) in the volume group. Any given LTG copy can, of course, have more than one write I/O in progress to it. Since optimum disk subsystem write throughput is achieved when a queue of several write I/Os is maintained on every physical disk, if write throughput is an issue and MWC must be used, a volume group should be only as large as is required to implement the desired striping and mirroring of those logical volumes in the volume group. Many small volume groups should be defined in preference to one large one.

From a database performance standpoint, enabling MWC on mirrored logical volumes can be quite costly, especially for those database files with high I/O activity, such as the redo logs. This is due to the fact that two writes are performed for every update: One for the MWC cache record and one for the actual data. However, the performance degradation associated with writes to MWC enabled logical volumes can be minimized through the use of hardware write cache. Certain SSA adapters, such as the IBM SSA Advanced SerialRAID Adapter, support an optional 32 MB non-volatile fast write cache.

If MWC is disabled to improve overall I/O performance to one or more mirrored logical volumes, some other measure must be taken to guarantee mirror consistency in the event of a system crash. In order to ensure that all copies contain the same version of the data, you must disable autovaryon for any volume groups that contain such logical volumes and manually force synchronization (`syncvg -f -l LVname`) of every mirrored LV. The `-f` option tells `syncvg` to choose a good physical LV copy and propagate it to all other copies, whether any copy is marked stale or not. Every `syncvg` must complete before varying on the volume group and making logical volumes available for use by the database.

MWC can be disabled to improve batch load performance provided:

1. The logical volumes involved are completely reloaded from scratch if the system crashes and reboots.
2. MWC is enabled as soon as batch loading completes.

Consider the following example:

In the middle of a database update transaction, a system crash occurs preventing some mirrored copies from being updated. The LVM does not have time to mark any partitions stale. The system restarts, and the LVM automatically varies on the volume group(s) containing the data in question. Because none of the partitions are marked stale, logical volumes are placed back online without undergoing any form of synchronization.

At this point, the database initiates some form of crash recovery to roll back any failed transactions. The first copy of the mirror (copyA) indicates that the data was changed, but the transaction did not complete. Another copy of the data (copyB) indicates that the data was not changed. If copyA is chosen, and the transaction is rolled back, everything will be consistent from a database perspective, as all copies will be written to during the roll back of the failed transaction. However, if copyB is chosen, no recovery is performed, and two different copies of the data still exist, which can cause logical corruption of the database.

The main drawback to disabling MWC is that manually resyncing the mirrors can significantly increase the amount of time necessary to make the volumes available, as every single partition must be read and rewritten to all of the mirrors. A decision must be made as to which is the most important: Speed of recovery in the event of a system failure or performance of the application.

8.8 Serial Storage Architecture (SSA)

Serial Storage Architecture (SSA) is an open-storage interface used to connect I/O devices and adapters to host systems. SSA was designed to be a high-performance, low-cost alternative to traditional SCSI based storage systems. SSA also directly addresses some of the performance and manageability limitations of the SCSI architecture.

8.8.1 Technology overview

SSA subsystems are comprised of *loops* of adapters and disk devices. A theoretical maximum of 127 devices can be connected in a SSA loop, although current IBM SSA adapters limit this to a maximum of 48 devices per loop. Each SSA adapter has two loops, each with two ports or *nodes*. Data transfer is bi-directional in a SSA loop, with a maximum data transfer speed of 40 MB/s in each direction, for a total transfer speed of 80 MB/s per node or 160 MBs/ per loop.

In SSA terms, a node can be either an *initiator* or a *target*. As stated previously, each adapter contains two nodes or ports, and each SSA disk device also contains one node. The SSA adapter nodes are the initiator nodes responsible for issuing commands to the target nodes on the attached SSA disk devices.

SSA provides the following performance and manageability advantages:

- Dual connection paths to attached devices - If a break occurs in a loop, the data is automatically rerouted.

- Simplified cabling when compared to SCSI - Cheaper, smaller cables and connectors, no need for separate terminators.
- Faster interconnect technology:
  - Full-duplex, frame-multiplexed serial links
  - Capable of transferring data at 80 MB/s per port, 160 MB/s per loop and adapter
- Hot pluggable cables and disks
- Supports large number of devices - Up to 127 per loop, although current IBM SSA adapters limit this to 96 disks per adapter.
- Auto-configuration of attached devices and online discovery.
- Increased distance between devices - Up to 25 meters with copper cables and 10 kilometers with optical fiber extenders.

8.8.2 SSA specific performance considerations

There are various performance factors specific to SSA implementations that must be considered when designing your disk subsystem. These include:

- The number of disks per SSA loop or adapter
- The distribution of the data among disks in a loop
- The position of the device in the loop

8.8.2.1 Number of disks per SSA loop or adapter

While the SSA adapter itself is capable of supporting a peak data transfer rate of 160 MB/sec, the host interface or bus usually limits the speed to a fraction of that supported by the adapter. The maximum sustained data transfer rate is approximately 90 MB/sec. Given that limitation and the maximum sustained data rates for different SSA drives listed in Table 11 on page 172, the actual number of disks that can be effectively placed on a loop or adapter is much less than the maximum of 48 per loop and 96 per adapter.

Table 11. SSA Disk Characteristics

Formatted Capacity (GB)	**1.1**	**2.2**	**4.5**	**4.5**	**9.1**	**9.1**	**9.1**	**18.2**	**18.2**	**36.4**
Media data rate (MB/s)	9.59-12.58	9.59-12.58	9.59-12.58	11.52-22.4	10.3-15.4	11.5 2-22.4	15.2-30.5	11.52-22.4	15.2-30.5	17.9-28.9
Sustained data rate (MB/s)	7	7	7	9	9	14	18	14	18	18
Average seek time(ms)	6.9	7.5	8.0	6.5	8.0	6.5	5.3	7.5	6.5	7.5

Formatted Capacity (GB)	**1.1**	**2.2**	**4.5**	**4.5**	**9.1**	**9.1**	**9.1**	**18.2**	**18.2**	**36.4**
Rotational speed (rpm)	7200	7200	7200	7200	7200	7200	10K	7200	10K	7200
Latency (Average)	4.17	4.17	4.17	4.17	4.17	4.17	2.99	4.17	2.99	4.17
SSA Transfer Rate (max MB/s)	20	20	20	40	20	40	40	40	40	40
Drive Type	DFHCC1B1	DFHCC2B1	DFHCC4B1 or DFHCC4C1	DGHC04B	DCHC09B1 or DCHC09C1	DGHC09B	DRVC09B	DGHC18B	DRVC18B	DRHC36B

The Drive Type information shown for each drive in Table 11 corresponds with the Machine Type and Model field shown in the output of the `lscfg` command when used to query the configuration of an SSA physical disk. For example:

```
# lscfg -vl pdisk5
  DEVICE            LOCATION          DESCRIPTION

  pdisk5            30-60-P           2GB SSA C Physical Disk Drive

        Manufacturer.................IBM
        Machine Type and Model......DFHCC2B1
        Part Number.................02L7551
        ROS Level and ID............9190
        Serial Number...............6811D707
        EC Level....................E29646
        Device Specific.(Z2)........RAMSC091
        Device Specific.(Z3)........02L7551
        Device Specific.(Z4)........97246
```

The number of disks that can be effectively placed on a SSA loop or adapter is largely dependent on the I/O characteristics of the application that will be accessing the data. The exact number of disks that will provide the most optimal performance will obviously vary depending on the workload placed on the disk subsystem by the application. With that in mind, the following general rules of thumb apply:

- If the application primarily performs long sequential I/O operations, a maximum of 8 to 16 disks should be configured per SSA adapter. This configuration would provide sufficient bandwidth to saturate the host system bus in a PCI architecture.
- If the application performs a mixture of sequential and random I/O operations, then 16 to 32 disks per adapter would be sufficient.

- If the application is characterized by many short transfers with random seeks, the chances of any one disk saturating the available bandwidth of the adapter or bus is fairly low; therefore, more disks should be added per loop/adapter. In this instance, two loops of 24 to 48 disks per loop should provide adequate performance while still staying within the limits of the adapter or host system bus.

8.8.2.2 Distribution of data among disks in a loop

The SSA architecture allows data to be transferred in different portions of a loop concurrently, a concept known as *spatial reuse*. The distribution of I/O operations to the physical disks in a loop must be carefully considered when designing the disk subsystem in order to take advantage of spatial reuse and maximum throughput on the adapter. As an example, consider the one adapter, single loop, eight drive configuration depicted in Figure 28.

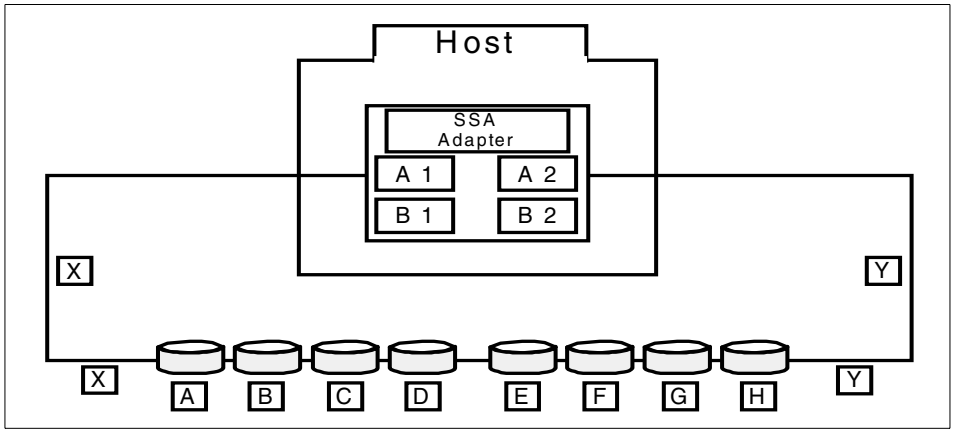


Figure 28. Spatial reuse in a SSA disk subsystem

With this configuration, the adapter accesses disks A, B, C, and D using portion X of the loop and disks E, F, G, and H along portion Y of the loop.

Due to the fact that the SSA link is full-duplex, each portion of the loop can concurrently process both read and write operations. Optimal utilization of the bandwidth of the loop would occur if, at any given point in time, 50 percent of the operations in portion X of the loop were reads and 50 percent were writes. If, however, 100 percent of the reads were accessing disks in portion X of the loop, and 100 percent of the writes were accessing disks in portion Y of the loop, only half of the available bandwidth of the adapter would be used.

Optimal performance can be obtained by ensuring that disk accesses are balanced across both portions of an SSA loop. This will ensure that both the

available bandwidth and the number of disks that can be placed in the loop are maximized.

8.8.2.3 Position of the device in the loop

The proximity of a device in a loop to the SSA adapter can have performance implications in heavily loaded SSA networks. In these cases, the SSA drives that are furthest away from the adapter consume more of the available bandwidth than those that are closer to the adapter.

In an SSA loop, data packets are passed from the originating device or node to a packet buffer on an adjacent node, and so on, until the packet reaches the adapter. If the SSA loop has many disks, each passing data packets to their adjacent nodes, the disk(s) nearest the adapter can become very busy passing data packets to the adapter that originated farther down the loop. In order to ensure that the devices nearest the adapter do not become stuck passing data packets for other devices on the loop, the SSA architecture employs a fairness algorithm that uses a token passing mechanism.

One token, known as the *SAT* token, circulates in each direction around the loop. If a device on the loop has outstanding I/O requests that needs to be processed, it must wait until it receives the token. The queued I/O requests are then processed in sequence up until a point in time at which the fairness algorithm determines that it must stop. It must then wait until the token recirculates around the loop before processing any additional I/O requests.

Even with the introduction of the token, devices nearest the adapter will still be responsible for passing more data packets to the adapter than those further down the loop. Therefore, in heavily loaded configurations, you should consider placing disks with the highest I/O as far away from the adapter as possible.

8.9 Integrated disk storage systems

Providing access to data in an open systems environment represents a substantial challenge given the multitude of servers and storage systems that exist within most organizations today. The ability to manage and share information is a pressing concern as administrative costs continue to climb, and the management of multiple servers and data storage systems grow more complex every day. These concerns, and the spiraling costs associated with them, have led many organizations to a strategy of server consolidation: The relocation of distributed servers and related storage into data centers with centralized management provided by corporate IT departments.

Companies need storage solutions that are flexible and designed to simplify storage management, provide sharing of storage resources, enhance data sharing, and support their plans for server consolidation. Storage consolidation can be the first step towards the goal of implementing server consolidation in reducing the number of boxes and providing IT departments with the flexibility to add or assign storage capacity when and where it is needed.

With the increasing standardization of Internet technologies and focus on Web-centric computing, rapid growth in global e-Business is increasingly demanding continuous computing and 7x24 access to data. Intelligent storage subsystems are emerging as key players in providing servers with the capability to off-load administrative tasks, such as copy services, and even executing backups without impacting the production servers. In response to this trend, IBM offers a series of products on the Seascape Enterprise Storage Architecture platform that includes the Enterprise Storage Server (ESS). The ESS's high performance, attachment flexibility and large capacity provides the ability to consolidate data from different platforms onto a single high-performance, high-availability box.

8.9.1 IBM Enterprise Storage Server (ESS)

The IBM Enterprise Storage Server provides universal data access including RS/6000 and SP2 running AIX, many leading UNIX variants, IBM Netfinity, and other Intel-based PC servers running Windows NT, Novell Netware, and AS/400 running OS/400. Any combination of these heterogeneous platforms may be used with the ESS including S/390. The IBM Enterprise Storage Server is a storage subsystem that is compatible with previous generation of IBM storage products. The following key design and functional characteristics define the ESS:

- The ESS provides connectivity to UNIX, NT, and AS/400 hosts through SCSI (32 SCSI-2 or SCSI-3) interfaces S/390 connectivity is provided through ESCON channels (32). A combination of ESCON and SCSI channel configurations provide multi-platform connectivity.
- Designed to handle a wide variety of open system hosts, the ESS provides these hosts with the ability to address 4096 SCSI and 4096 S/390 devices through 16 host adapters, each supporting two ports. With support for the SCSI-3 protocol, the ESS offers the assignment of up to 64 logical unit numbers (LUNs) per SCSI target or SCSI ID, thus providing a maximum of 15 x 64 or 960 LUNs (one host adapter used by the host). While not all open host systems support the full 64 LUN capability, AIX handles up to 32 LUNs/SCSI target, while NT only handles 8 LUNs/SCSI target. For UNIX and NT systems, the IBM Data Path Optimizer can be used to

provide multiple paths (up to 16 paths) from a single host to shared LUNs, thus, providing ways to distribute the workload in order to avoid bottlenecks.

- Fibre Channel connectivity to open system hosts is provided through the IBM SAN Data Gateway product. In the future, the ESS will support the native Fibre Channel Protocol (FCP), Fibre Channel Arbitrated Loop (FC-AL), and FC-switched capability for all open systems, while Fibre Channel (FICON) protocol will support S/390.
- The ESS contains two, independently-managed, 4-way RISC (332 MHz) SMP processor clusters connected to each of the host adapter delivering leadership fault tolerant storage subsystem functionality. Failover and failback provides redundancy for the disk arrays connected to both clusters.
- Each cluster is supported by 3 GB of non-shared cache (total of 6 GB) to store both READ and WRITE data to improve performance to attached hosts system. In the future, this cache will be increased to 16 GB in total.
- Each computing cluster in the ESS also has 192 MB of battery-backed, non-volatile storage (total 384 MB) to store a second copy of WRITE data to ensure data integrity in the event of an unexpected loss of a cluster, power failure, or cache copy loss. The design and process of failure management in the ESS ensures that no data is lost even in the event of component failure.
- Using the latest SSA 40 MB/sec full-duplex loop based interface technology within its 8-pack disk array, and with its four pairs of device adapters, accessible by each of its two independent clusters, the ESS provides a total internal disk bandwidth of 1,280 MB/sec.
- The basic unit of capacity in an ESS is the 8-pack or a group of eight disks in a drawer. These disk arrays are called "ranks" in the ESS. RAID rank arrays can be configured as RAID arrays or as non-RAID arrays known as Just a Bunch of Disks (JBODs requiring only one disk to form an array). A RAID5 rank array of eight disks can be configured either as (6+Parity+Spare) or (7+Parity). Using 36.4 GB disks (7200 RPM), a single ESS with an expansion rack delivers approximately 11.3 TB of usable capacity. Other available configurations options include 18.2 GB disks (10000 RPM) and 9.1 GB disks (10000 RPM). For investment protection, the ESS also supports 7133-D40 drawers and 7133-020 drawers when the 2105-B09 VSS or 2105-100 VSS racks are attached to it.
- The ESS provides several hardware-assisted advanced functions for delivering instant copy, mirroring, backup (including split mirror backup), and disaster recovery capabilities for UNIX, NT, and S/390 environments:

- FlashCopy delivers the ability to make near-instant (called time zero - T0 copy) identical copies of specified source LUNs/volumes to target LUNS/volumes. The target volume, or copy, is accessible for READ or WRITE immediately upon execution of the selected FlashCopy command. This enables applications to be stopped for a very short period of time, while instant backups can be performed using FlashCopy.
- Peer-to-Peer-Remote-Copy (PPRC) is a robust, synchronous, IBM-patented copy function that provides mirroring (RAID1) capability between two ESSs connected by ESCON links (fiber optic links using S/390 ESCON protocol). Currently, PPRC functions over a distance of 103 kilometers using third party channel extenders.

- For ease of management, the ESS comes with its own resource management tools provided by the StorWatch ESS Specialist. Accessible through any Web browser that supports Java 1.1, the ESS Specialist must be used to configure the ESS, administer Copy services (through the ESS Copy Server), set up FlashCopy and PPRC functions, and partition ESS capacity among attached hosts.
- In the future, IBM plans to deliver a virtual storage capability through the implementation of the Log Structured File (LSF) architecture currently available only on IBM RAMAC Virtual Array (RVA product).

8.10 Disk performance measurements and observations

In order to validate some of the assumptions and recommendations presented in this chapter, we performed a series of tests using the model database described in Appendix D, “The Model Database used for testing in this redbook” on page 401. These tests were separated into two different categories:

- Evaluation of database performance on raw logical volumes versus JFS file systems
- Comparison of different data placement strategies: RAID 5, JBOD + mirrors, and mirroring and striping.

For the evaluation of raw devices versus JFS file systems, we used the test queries described in the Appendix D to simulate both OLTP and DSS workloads. The OLTP queries consisted of a mixture of read, update, and insert operations, while the DSS queries were comprised of 100 percent read operations. Based on these tests, raw logical volumes were approximately 15 percent faster than JFS file systems in an OLTP environment and approximately 35 percent faster in a DSS environment. The rather large

performance advantage associated with using raw logical volumes in a DSS environment can be attributed to the increased sequential read activity that often takes place in DSS systems.

Note

In our tests, raw logical volumes were faster than JFS files:

- 15 percent for OLTP
- 35 percent for DSS

These results agree with comments from the IBM Austin performance experts, but the IBM Austin performance experts comment that the difference in performance is not the result of the double buffering used by the JFS buffer cache (faster CPUs and CPU caches have reduced this to a minimum) but due to the locking requirements when using a file system to access disk blocks. If an RDBMS used the Direct I/O feature, it could lead to an increase in performance for JFS based databases, but the effect is very small when compared to the performance limits caused by the inode locks. In practice, the RDBMS vendors have not implemented Direct I/O because the performance gains do not warrant the development and testing costs. For faster disk I/O, they recommend raw devices.

The performance data outlined in Table 12 is the result of a series of tests that we performed simulating workloads typically found in OLTP and DSS environments. AIX Version 4.3.3 was used in order to take advantage of the new LVM striping + mirroring capabilities. The disks were attached to an IBM SSA Multi-Initiator/RAID EL Adapter and used raw logical volumes.

AIX 4.3.3 mirroring + striping

Logical volumes, configured with the new AIX 4.3.3 mirroring and striping functionality option, were tested as part of this redbook and performed well.

They can be recommended for databases.

Table 12. Relative disk performance in OLTP environments

	Relative performance 5 percent write operations	Relative performance 10 percent write operations	Relative performance 20 percent write operations	Advantage	Disadvantage
JBOD	80[1]	80	80	Low cost	No disk protection and manual effort
JBOD + mirror	120	110	100	High speed	Manual effort[2] Costs
4.3.3 striping+ mirroring	110	100	90	No manual effort	Slower than JBOD + mirror

Notes:
The test scenario was:

- Random I/O on multiple 1 GB files
- 4 KB reads and writes
- High concurrent access (16 parallel I/O requests)
- JBOD = 4 disks, Mirrors = 8 disks

[1] The relative performance number serves as an overall indicator of the performance of the particular data placement option (JBOD+mirror, and 4.3.3 striping and mirroring) relative to each of the other options.
[2] Manual effort refers to the process of manually distributing the database's datafiles among the disks and their mirrors in a JBOD configuration in an attempt to balance the I/O load of an application. LVM striping automatically distributes the I/O load across multiple physical disks through the use of data striping techniques.

DSS workloads exhibited roughly the same performance numbers. However, AIX 4.3.3 striping + mirroring performance numbers were slightly better due to the increased sequential read activity found in DSS applications.

Note

The performance numbers expressed in this section in no way represent a definitive statement as to the maximum or minimum capabilities of a particular disk configuration. They merely serve as a relative indicator of the performance observed in the specific environment in which they were tested. No hardware or software specific performance tuning was attempted during the course of these tests. The SSA adapters used in these tests did not possess the Fast Write Cache option that is available for the most current SSA Advanced Serial RAID adapters.

8.11 Choosing your disk subsystem

Based on the experience of designing disk sub-systems, performance tuning production databases, and the performance tests conducted as part of the redbook, Figure 29 is a summary of the process of choosing an appropriate disk.

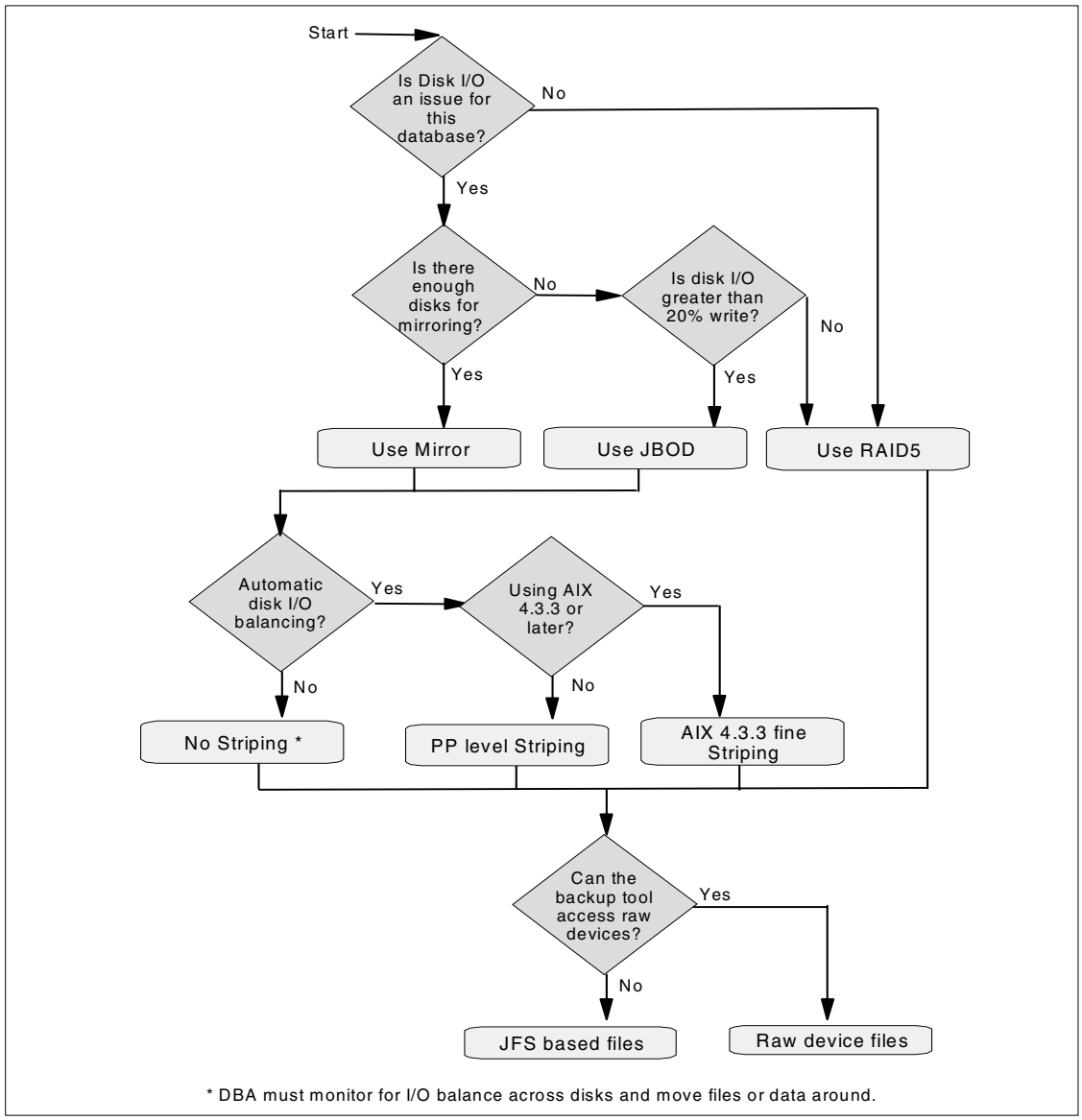


Figure 29. Choosing a disk subsystem

There are three levels of decisions that need to be considered:

1. Disk protection
2. Striping (if it is not RAID)
3. Whether to use the file system or not

You will need to decide on your priorities. All the possibilities from this diagram are expectable, but two extremes might be:

- For low maintenance, low cost with disk protection, and low I/O rate databases, RAID 5 with JFS is a good choice. However, the fast write cache enhancements on the latest SSA RAID adapters improve I/O performance, especially write performance, significantly.
- For high performance, disk protection, and I/O bound databases, mirrored and fine striped AIX 4.3.3. raw devices are a good choice.

There is no perfect choice.

Part 3. System optimization

Chapter 9. Implementing your database

This chapter covers hints and tips on how to install, set up, load, document, and test your database system. This is basically a chapter to let you learn from the mistakes of others, and the hints have been developed from experience of large production systems, especially benchmark systems. Benchmark systems are created from scratch for each benchmark and often on new hardware and software versions; so, a lot of experience is developed in a short time. The aim is to make your installation as simple and straight forward as possible and to avoid common pitfalls.

This chapter does not cover how to install the RDBMS code itself. This is very different for each RDBMS and even changes between releases. For details on how to install your RDBMS, please refer to the documentation supplied with the product.

Each RDBMS installation process has its own quirks. For example:

- Oracle, from 8.1.5 on, no longer requires the Oracle 7 `start.sh` script to be run to create thousands of symbolic links to the contents of the CD-ROM to make the files on the CD-ROM readable by the installation application *orainst*. Oracle now has a graphical user interface tool to guide the DBA through installation; so, access through an X Windows console, X Terminal, or X Windows emulation on a PC is now advisable.
- DB2 UDB, on the other hand, loads like any other IBM licensed program product (LPP) but is large because it comes with all the online documentation for every supported language. This makes it a long process to copy the images between machines for remote installation over a network. A recently ordered complete set of DB2 UDB installation code came on 46 CD-ROMS, although that did include many samples and *try before you buy* CD-ROMs of DB2 support and development programs.

Most people rush into installing RDBMS code before really being ready. The pressure to start loading is clear and understandable. The machine has been delivered; the project has been waiting for some time; the various parts are ready, and everyone on the team wants to see some real progress before they go home. But, this pressure should be resisted. It is like a race car driver arriving at the race track for the first time and is itching to drive around the circuit. The driver will, of course, want to drive the race car, but it is still being prepared and tuned. As an alternative, the driver could take out a standard road car to *have a go,* but it is clear that the results, although interesting, will not help in the effort of winning races. The same is true of installing the RDBMS without the necessary planning and care. Indeed, many hours or

days might be lost because what should be a quick test might fail to work, and it might take a long time to track down the cause. Worse still, a lot of damage can be caused by rumors and panic about this project being a failure because of reports that the machine and RDBMS do not work.

.

Note

The recommended implementing approach is:

- Do it once.
- Do it right the first time.
- Know exactly how you did it in case you have to do it again in an emergency.

The rest of this chapter will give you guidance to achieve this from the experiences of people that do this all the time and have learned from hitting the problems themselves.

9.1 Hardware and AIX ready check list

The requirements before installing the database depend on who you are, or more precisely, your job role. If you are the AIX system administrator, then you are responsible for installing the RS/6000 in the first place. As an alternative you might be the database administrator and be using an internal I/T infrastructure team, IBM global services, a facilities management company, IBM business partner or consultants to do this part of the work.

Whoever is preparing the RS/6000 hardware and operating system needs to have the following ready and running:

- Basic machine hardware running AIX.
- Recent version of AIX. The AIX code is (very approximately) updated once per year. The same is true for RDBMS versions. You are strongly recommended to keep up with the latest version to enjoy full and prompt support from your vendors. As this is a new system, make sure you at least start off with the current version of AIX and save the effort of upgrading shortly afterwards. The only reason for installing an earlier version is that application support is some times lagging behind the latest version but only by one release. Also, from experience, both AIX and the RDBMS have improved performance with each release and additional new functions that either give performance or reduce the manpower required to maintain the system. For example, AIX 4.3.3 has striped and mirrored logical volumes (partitions) and enhanced disk I/O performance. Or with

the RDBMSs, both Oracle and DB2 UDB have improved SQL optimizers for better SQL query plans and have added object orientation support and Web access in recent versions.

- Set up the AIX *root* user with a sensible password. This will need to be changed later once the system is ready for production use. Therefore, use something easy to type quickly for now (not something guessable, such as the machine or project name or persons name) but do have a password. During the set up and installation, many people might know the root user password to configure the various parts of the system. A simple password does this and will not waste time. Do not use a password with odd upper and lower case characters or passwords that are meaningless and easy to get wrong. Once every thing is working, root access should be limited to only those that really need root authority.
- Change the maximum number of users to the licensed limit. When the system was purchased, this included a particular number of users. It is simple to forget to install the machine with this number set up. To do this, use SMIT and use the *System Environment* menu. This is one of the few things that requires a system reboot; so, get this done now. The two user licence default limit is too low for installation, and you will find the machine refusing user logins; therefore, it can take some time before you workout what is causing the problem. Note: If you run into this problem, the *root* user can still log in, and you can then `su` to the user you need.
- Increase the maximum number of processes per user and set this to a number sensible for the RDBMS user who will be creating many of the processes on behalf of the users. This limit will stop the RDBMS from handling a lot of users and cause problems on the first test of the system with more than 30 to 40 users. You do not have to reboot the system, but you will have to stop the RDBMS completely, log out, and then log in and restart the database.
- The paging space created and online. There is a large debate on the size of paging space to use. We recommend that the absolute minimum is one times the size of memory. Most people would argue for much more than this. It might seem a lot when you have 32 GB of memory, but below this, you are taking a serious risk. This is a classic benchmark system mistake to forget if a quick test is run on the default paging space. If this test requires a lot more paging space, then absolute mayhem is guaranteed. Processes start crashing while running (when requesting more memory), new processes (when forking) fail, and AIX automatically attempts process re-forking at 30 second intervals. Also UNIX commands fail to start, even root cannot log in, and usually you can either wait three hours and AIX will eventually work through these unfair demands and recover, or you can halt

the machine with the reset button and wait for the long, full system check reboot. It is worth avoiding these problems by spending ten minutes setting up a proper sized paging space. Refer to 7.1.1, “Basic and future AIX resources” on page 121 for more details.

- Have all the media devices of the system working, such as the CD-ROM and tape drives, because they will be needed shortly. It is better to get these devices working beforehand, as this might require changes to SCSI cables or stopping the system to check the hardware. A good way to test the tape drive(s) is to create a system backup (using `mksysb`) of the initial AIX system. This could also save time if the installation of the RDBMS fails, and you need to clean up quickly. For example, incorrectly setting the owner and permissions in the /dev directory (required for raw device access by the RDBMS) could take hours to fix by hand. A simple mistake on the command line is all it takes.
- All databases, of course, require disk drives. Large databases require a large number of disks. All of the disk drives have to be connected and available. We assume that, in the hardware planning stage, a suitable design has been carried out on the layout and the configuration of the disks and adapters, for example, the number of disks that may be attached to SCSI adapters to provide full access to all disks at high speed. Additionally, for SSA or FCAL connected disks, the number of disks per loop and the loop design for recovery must be carried out. It is vital to check that the documented design has actually been implemented. It is very easy for hardware to be configured wrong by mistake. Or, the hardware people might have made what they thought were improvements to the design without telling anyone. It is important to have a very clear, full, and documented understanding of the adapter, loop, and disk configuration.
- Multiple user access. Do not try implementing an RDBMS on the only dumb terminal connected to the machine. Often, in the course of setting up the machine, you will want two (or more) screens, as you will be partly through one activity, and you will want to check something without stopping the tool. Two screens, or better still, an X Windows access to the machine, will save you time and frustration.
- Hardware sign off. Get agreement with the hardware engineer installing the machine that the machine is complete (all parts delivered) and that everything is working and there are no outstanding activities. On large machines, this can take time, and you may have been told the machine is working, but you were not told that some final part or cable is missing or that not all of the hardware installation process has been completed. It is best to double check that all is finished. Also, make a check before the

hardware support personnel go, as it is easy to miss things, such as earthing cables on metal covers and that all the cables are neat and tidy and their weight supported correctly.

- Finally, check that the machine reboots cleanly and without user intervention.

Summary

- Get the machine running
- Install the latest version of AIX
- Set the `root` user password
- Set the user licence limit
- Set the processes per user
- Set up paging space
- Quick test the devices, such as CD-ROM and tape drives
- Quick test the disks and ensure that they are connected appropriately
- Use an X Windows terminal
- Get hardware sign off
- Check the machine reboots cleanly

9.2 Pre-starting check list

The last section covered the basic hardware and AIX, but before starting to install the RDBMS, there is another list of things to check and information to have at hand. These include:

- The AIX installation media. You may be told to load extra features of AIX to support the RDBMS, the application, or administration tools. Do not waste time trying to track these down later; so, make sure they are available and are the correct version.
- Access to AIX manuals. You must have some means to access the AIX manual pages and AIX information. This could be hardcopy physical manuals, but it is more often the case these days to be the online manuals. Make sure they are current. Some of the system administrator commands have new options on each release, unlike the normal user commands that are all now POSIX standard and change little between releases. Also, have the hardware manual for the actual machine, the disk drive subsystem, and the tape units you are using, along with the RS/6000

product documentation with all the boot LED codes in case there are hardware problems. Again, this will save you time later.

- The RDBMS installation manual. The number of people attempting to install a database system without this is amazing! We have already said there are often subtle changes between versions that will catch out anyone assuming it will be the same procedure as the last version. Your notes from the last install might be acceptable for a test or development system but not for a production system.
- The RDBMS readme files will include last minute information that the vendor thinks you need to know before installing. So, make sure you have it and have read it.
- The application installation media. This might be a third party application on CD-ROM or in-house binaries supplied by your own development team. It is particularly true for in-house development that the application is often not clearly documented or controlled. Many times these things are left until the last minute, and the development team has trouble getting the function and features correct before sitting down to write documentation about things, such as installation procedures.

  Whatever the source of application, you should know:

  - Precisely which version it is
  - What level of testing it has been through
  - If optional parts are required or not
  - Any dependencies on AIX level or extensions, RS/6000 architectures, and RDBMS level and extensions
  - The required size of both disk space and memory per user
  - Placement recommendations in the file systems
  - Pre- and post-installation requirements, particularly if the RDBMS needs to be modified
  - Configuration parameters
  - How updates are going to be handled
- The RDBMS vendors and application recommended tuning parameters and options for the RDBMS. These are likely to change with experience of running the system, but you need to start somewhere.
- The database data. See next section

9.3 Database data

We are covering the database data at this point because most of the problems with database data can be tackled before the system is ready for use. At this point, you can gather the information and make sure the data will load once the database is ready. As databases grow in size, it is often forgotten that handling large volumes of data takes time. If the data is found to be wrong when it is being loaded, it can take hours or days to prepare an improved version. It might take weeks if it has to be fitted into a production cycle.

Particular care must be taken regarding the database data. Very few databases start with no data at all. We assume the application install will create the empty tables and perhaps the indexes. Some applications hide all of the complexity of the database from the installer, but this requires a lot of work from the application provider. It also limits the choices for the installer. If this is the case, you have to have faith in the application vendor or take a look at the information to determine what has been pre-decided and if it is going to perform well on your system. Other applications tell the installer to modify and then run a database creation script. Either way, there are a number of questions that need to be asked about the actual data to be placed within the database once created. Also, note that precise answers are required. If you get either imprecise answers, or are told not to worry, then start worrying. Here are the basic items you need:

- The database schema describes the objects and relationships in the database. All good databases or applications should have a schema diagram available. The objects are implemented as tables, and the relationships are used in SQL to join tables together (in the `WHERE` clause). The database schema is often a large entity relationship diagram, as this is a good way to represent the schema's objects/tables (by boxes on the diagram) and relationships (by lines joining the tables). Some databases and applications will simply have the table creation script. For a simple example, see Figure 44 on page 402.
- Referential integrity policy. Many applications use the SQL standard to implement referential integrity using primary keys (the RDBMS will use an index to enforce this behind the scenes) and to define the other tables referring to this key. This ensures that a row referring to the primary key of another row in another table actually exists. For example, if an employee row states the employee works in department 42, then there must be one row in the department table with a primary key of 42. The script to create the tables will include these keys, and their use is explicitly described in the SQL. But, note that primary keys and enforcing them does create an

overhead and has a performance cost; so, many applications do not enforce this on production databases. In this case, you should be provided with a script or program that can check for referential integrity.

- A list of every table, its definition, the average row size, row count, and size including the RDBMS overhead (to allow for rows being placed in disk blocks and some wasted space at the end of each block). This is often in the form of a spreadsheet to allow simple adding up of all the various columns to work out the overall size.
- A list of every index, its definition, uniqueness, average size, and size including RDBMS overhead. This also is often in the form of a spread sheet to allow the simple adding up of all the various columns to work out the overall size. Indexes are vital for high performance, and a tuned application will demand a minimum set of indexes, while others might be added after production goes live and are based on observed access patterns.
- The tables and indexes that are very busy (some times called the hot tables). These are the items that are critical for the best performance of the system. They will need careful placement on the disks of the database to ensure that overworked disks can be avoided. They can be placed on dedicated disks, or the data can be spread across many disks to ensure I/O bandwidth.
- The data source. If this is a machine that is very different to the database machine, then problems can be expected. If it is another UNIX machine, it is worth checking the tape media are compatible, and the tape commands use the same format. It is worth actually trying if a small sample can be loaded. An alternative is using a network, but check that it can handle the data bandwidth requirements and actually works. If it is a different architecture, like IBM AS/400 or IBM mainframe, then both the tape format and the data format is worth checking. Extra care needs to be taken if the volume of the data means multiple tapes because *end of tape* marks and formats are often a difficult problem. Generally, the UNIX `dd` command can load the raw data from any tape drive, provided the drive recognizes the format and tape density. But, you may need a very good C programmer to extract the data from the file that was read from tape and produce a file that can be loaded into the RDBMS. If there is any doubt, run a test beforehand.
- The date when the data is available and if improved data is available at a later date. Many systems are tested with early data and then reloaded again with the final data just before going live. This might be because the data manipulation and cleaning is not finished or because the production

system needs to go live with the very latest copy of the data, as it will have been updated during the test period.

- Verify that the data has been test loaded on the same RDBMS version. Developers often assume later releases will work the same, but there can be subtle differences.
- The instructions for loading the data. Many data suppliers assume you can read their mind and do not supply basic information. Is it clear which data is for which table? Is it clear in which order to load the tables? Is it clear which tape contains what data? Many companies have tape defaults and standards and assume everyone else understands these. Benchmark centers are often sent tapes labeled TAPE1 to TAPE9. It takes time to work out the command used to create the tape. It takes time to work out the block size. It takes time to work out the options to the command used. Then you can start to work out what data is on the tape.
- Maintaining referential integrity during the load. If the SQL standards for integrity are used, it can be very difficult to load a database because rows and tables have to be loaded in the right order, and this makes using bulk loading methods very hard to use. It is easy to bulk load 100 million rows, and only afterwards notice they were all rejected because they refer to a table that has not been loaded yet. The alternative is to disable (sometimes called switch off) referential integrity while loading the data and then enable it afterwards. Once the database is loaded, it is important to check it was loaded correctly. The absolute minimum is to count the rows of each table. This makes sure that it was correctly loaded and ensures that the table has not been forgotten or loaded twice. Also, check that the indexes are built-in and valid.
- Recommended load method. Each RDBMS has various means to load data including simple, bulk, and very fast loading methods. If the data for a particular table is large, then it is important to load it using the right method, or it might take days or weeks to load the database. For example, on one benchmark, the simplest method (Oracles PL/SQL) was used to create a sample database, and it worked well. But when it was rerun to create the full database, it seemed to be taking a long time. On checking the progress, it was worked out that it would take three and a half weeks to complete. A better approach was developed using a C program to generate the raw data, which was then loaded using the Oracle loader.

This might seem to be a long list of items, but this aspect of the database implementation process is often ignored, postponed, or covered at the last possible moment, and the complexity is often not appreciated. A lack of planning can bring the installation of a database system to a complete stand still. Planning ahead and having the right data and information at the right

time can often reduced the time it takes to install a system. A lot of the preparation can even take place before the machine arrives.

9.4 Hardware testing

Before you start loading a large amount of data onto a machine, a few things about hardware should be acknowledged. First, the good news:

- Computer systems get more reliable every year.
- Each component is designed to self-check. For example, memory with ECC and parity will automatically check for memory errors and even correct them, if they are found, without stopping the system.
- The mean time between failures is larger too. For example, the mean time between failures on disks is longer with each generation of disk, and each generation of disk supports more and more data.

Indeed, IBM is committed to improving quality in all products and does testing to ensure each new product is more reliable than the one it replaces.

Now, the bad news:

- New systems get bigger every year.

The problem can be highlighted by an extreme example using made-up numbers to make the point: Product reliability over a five year period is doubled (that is, it fails half as often) but the system is now 10 times more complex (that is, 10 times the number of parts). So far, so good. But, the outcome is that the whole system will fail five times more often.

This is a very real problem for manufacturers of very large systems. Larger systems need more and more designed-in features to cater for failures in the various subsystems that can monitor and correct temporary and permanent failures and yet do not bring the system down. IBM is leading the way in this field in machines, such as the RS/6000 S Series, and is able to do this because IBM has such a large amount of experience in designing very reliable system for mainframe customers.

This means we must do everything possible to try to eliminate failures in components and, therefore, reduce the risk, however small.

Analysis of failures in large systems, such as large RDBMS systems, reveal three important facts:

1. Most failures happen quite soon, within the first 24 hours of serious use.

2. After that, failures seem to happen at completely random intervals but usually much later in terms of weeks, months, and years.
3. Occasionally failures happen in small batches. This sounds odd, but it happens. Take as an example: If you replace all the light bulbs in a house at the same time, you can notice several light bulbs will fail in the same week about a year later!

This is particularly true of hard disk drives because they have practically all the high speed moving parts of a modern computer system. CD-ROM and tapes have moving parts but not under the same constant pressure as disks. To eliminate disk failures, we must:

1. *Burn-in* the system - Run a little test on the system for 24 hours, particularly on the disks. It might take an hour to create some data on disk. Then, write a script to copy the data back and forth, but it will be worth it in the long run. This test does not have to be complex.
2. Prepare for failure with some form of disk protection, such as mirrored disks or RAID 5.
3. If you have protection via duplication, make sure they are as far away as possible from each other. For example, that disk mirrors are not in the same disk tower or drawer, not on the same loop or cable and not on the same adapter.

This might seem like overkill, but the big benchmark centers, who regularly install a 10 terabyte disk subsystem, do this because they have learned from experience. This *disk burn-in* will, of course, also test everything else, such as adapters, memory, buses, and CPUs.

Note

The larger the system, the more valuable a burn-in test will be.

9.5 Installing the RDBMS code

There are many explanations to the term *RTFM*. We prefer - *Read The Flaming Manual*. In this case, the manual is the installation manual or installation guide and the *read me first* information.

We advice following the RDBMS vendors standards and recommended procedures in all cases.

For example, Oracle recommends the Oracle Flexible Architecture (OFA) layout for the RDBMS code, scripts, and parameter files. Also, follow the

recommended names for the RDBMS owner names and directories. This makes it far easier for others to work on your system, such as IBM, Business Partner or third party consultants. These recommendations from the RDBMS vendors has been found to work by experience. Do not invent your own way of doing things, or you will be creating a problem that has already been resolved.

The recommendations also cater for the long term. For example, when you upgrade to the next release of the RDBMS and have to have two versions running at the same time.

It is very important to load all the recommended pre-requisite code, checks and AIX PTFs. It might seem like extreme caution, but they are there for a reason. This might be:

- Other customers may have had problems that you can now easily avoid and save time.
- If you have a problem, this will be the very first thing you will be forced to check and ensure it is correct by your support people or the vendors.
- If there are problems, and you have not done this, you will look, at best, very unprofessional.

So, install the RDBMS by simply and methodically going through the installation steps, one at a time, and make sure you do it right the first time.

9.6 Physical layout of the database

We are now at the point where the RDBMS is installed, and the next step is the creation of the actual database. We have to assume, at this point, that the database was designed, and this was documented. This documentation should include the placement and sizes for every part of the RDBMS system including:

- RDBMS code (actually already loaded at this point)
- Application code
- New versions of RDBMS and application and their fixes
- System catalogue or data dictionary
- Data
- Indexes
- Sort or temporary areas
- Logs (roll forward and backward)

This should also be summarized in a *data placement policy* so that if you need to make small changes during the database creation, you know how to keep within the design goals. Later growth and increases in the number of disks can follow the same principles. This will include:

- Disk protection used - This might be different for the various parts of the database.
- Naming conventions for volume groups and logical volumes.
- conventions for separating data (such as mirroring) and mixing data and indexes, or not.
- Strip sizes and the sizes of logical volumes.
- The number of disks in volume groups or RAID 5 LUNs.

You should also have the default RDBMS parameters to be initially used and options for changing these once some experience of the database is used. For example:

- The size of data blocks to use in the database
- How much memory has been allowed for in the RDBMS disk pool or cache and the other parts of the shared memory
- What levels of parallelism should be set and how this is set

9.7 Scripting the build

There are three methods for setting up the disks, installing the application, and loading the database data:

1. *Hacking* the machine by sitting at a terminal until it is done
2. Writing and running a script
3. A mixture of the above

Experienced people only use one method, and that is method 2 - scripting.

Note

Use scripts to create everything.

It is very tempting to just start and work your way through to the end, but please, just do not do it.

Take the time to place all the AIX commands in a script for things, such as:

- Creating AIX volume groups or RAID 5 disks
- Creating logical volumes and their stripes
- Adding logical volume mirrors
- Creating accounts and setting up `.profile` files
- Changing ownerships and permissions
- Creating journal file systems

Also, script RDBMS tasks, such as:

- Creating the database and loading the catalog/data dictionary
- Creating containers and tablespaces
- Creating tables
- Loading data
- Creating indexes
- Checking the database consistency

This scripting has a number of benefits that will save you time in the long run and has hidden benefits too:

- Scripting makes you think and plan ahead.
- Scripting makes sure everything is done in the right order.
- Scripting is actually faster - The machine does not have to wait for you to think of the next task or wait while you are gone for lunch.
- Scripting documents as to how it was done, which can be vital for disaster recovery.
- Scripting is much less error prone, as you can visually double check the commands and options.
- Scripting helps because, if it fails, it is simply a quick edit to get it right the next time. Even if you forget something major and have to start all over again, if it is scripted, you just fix the script and let it run over night.
- Scripting makes things consistent, such as naming conventions and sizes. If you are typing in all the names at the command line, it is easy to make simple typing mistakes that initially go unnoticed.

Once you have a set of scripts like this, you will find creating the next database much easier.

9.8 Build a small cut down system

If you have a database with more than 10 GB of data, then we advice first implementing a small cut down (say 1 GB or 2 percent, whichever is the greater) system as a quick trial of the main system. This allows you to:

- Work out how long the production system build will take. Particularly, the time to load the very large tables and indexes and investigate alternatives and options to reduce the load times. You might decide to use other tools if the time load is too slow.
- Sort out the scripts so that they will work the first time.
- Check the data formats are correct and readable.
- Check you have a complete set of data (although you might have to load a subset of the main large tables).
- Check the data is consistent and that you have tools to prove it.
- Check the application is complete.
- Check the application performs reasonably well even on smaller data sets and tests the indexes help performance.
- Check the administration and support tools actually work.
- Check the backup is complete and recovery can take place.
- Check the interfaces to the users, other application, and other computer systems work as expected.

This cut down system might be a throw away prototype system or might live on as a simple test system. Either way, the facts and experiences learned will be invaluable when it comes to building the full size system and should avoid common mistakes in handling large data volumes.

9.9 After installation

Every one promises to do these three things, but only the really smart ones actually go ahead and do them:

1. Finish the documentation and have a copy stored off site so that you are ready for a disaster.
2. Write a two page summary of what was learned during the installation, what went right, what went wrong and how to avoid the same mistakes the next time, what still needs to be worked on, and pass this information around the whole team. This is also called quality feedback to improve the process.

3. Start an RS/6000 system and RDBMS log book to record changes. This can be either on the system or hardcopy. This would mean that you always know what you have and why and when changes happened. This is invaluable for performance tuning later on and diagnosing the cause of problems.

> **Note**
>
> Start a system log book now.

9.10 Backup and recovery test

Once the database has been established and the data is loaded, the backup and recovery plan is very important.

You have taken some time in getting the database ready for use; so, save this work via the backup and, thus, ensure you do not have to repeat this work all over again.

Many sites do not test the recovery plan. When they then have a problem, it is too late. We all know a few horror stories:

- The tapes were not readable.
- They did not have the scripts to create the database and, therefore, could not reload the database.
- They were just about to recover the data from tape when the automatic backup script started and overwrote the backup with the corrupted database.
- They forgot to back up the logs (or some other vital database component); so, the backup was useless.
- The expert was away on holiday, and there was no one else who knew how to recover the data or even the tape format and commands to use.

Try hard not to be the starting point for the next industry horror story and check those backups and disaster recovery plans.

Documenting and recovery planning go *hand in hand*:

- Save the scripts used to create the initial database. They are a good start for writing recovery scripts and documentation for recovery.
- Automate the generation of every configuration detail into a regular report, such as the disks' layout and parameters.

- Store off-site copies of the details along with the backups in case the site is destroyed.
- Database backup is useless unless you can re-create the database into which to load the data first.
- It is extremely easy to forget vital facts and parts of the system; so, if in doubt, document it twice.
- Testing recovery is the only way to prove it - Schedule a recovery test once per year. At the very least, have an independent person check that the procedures, information, and tapes are readable.

And, finally, a warning: 70 percent of companies that fail to recover their IT systems within two weeks of a site disaster go bankrupt. Do not let your IT department be the cause of your company failing.

Chapter 10. Monitoring an RDBMS system for performance

Monitoring the RDBMS provides the database administrator with information about the interaction between user applications and the RDBMS and also between the RDBMS and the operating system. These indicators can help the database administrator to identify possible bottlenecks and to determine how to adjust the different RDBMS configuration parameters.

10.1 RDBMS tools

The DB2 UDB and Oracle performance monitoring tools can be used to assist an analysis of how the design of your system is affecting the way the database is performing. The database administrator should be aware of the most important success factors of the system, regarding not only how the system resources are being consumed but also how the database optimizer chooses the best and cheapest path to the data.

In addition to direct API and SQL interfaces into database statistic information, both RDBMSs include pre-defined GUI (Graphical User Interface) tools designed for easier database administration of both local and remote database instances.

10.1.1 DB2 UDB monitoring tools

DB2 UDB features two types of database monitoring, defined as follows:

- *Snapshot monitoring* - Used for obtaining database relevant statistics at a specific point in time[1]
- *Event monitoring* - Used for obtaining database relevant statistics over a period of time[2]

There is also a tool called *Performance Monitor* that uses the main functions of both the Snapshot and Event Monitor tools. The Performance Monitor is part of the Control Center.

Besides these monitoring tools, there is a tool called *Explain* that allows the database administrator to comprehend how the database is accessing the data.

No matter which tool you use for monitoring, it is necessary to keep in mind that the monitoring process is cyclical and should be part of the database administrator's everyday tasks.

[1] The interface to the snapshot monitor is through the use of APIs.
[2] The interface to the event monitor is through the use of SQL.

10.1.1.1 Snapshot monitoring

For snapshot monitoring, there are eight *levels* of data available. Six monitor *switches* are used to control the focus and the amount of data to be collected. These switches can be turned on and off dynamically to reduce the amount of resources dedicated to monitoring if the system is running unmonitored for an extended period of time.

Through the snapshot APIs, it is possible to collect the resources being allocated at the database level, drilling down to the tasks being executed by each application connected to the database. This is the fastest and easiest way to collect vital database information at runtime, such as all the resources being locked by a specific session, the amount of memory space used for sort operations, how many users are connected to the databases, and which statements are being issued by each connected user.

The following lists describe the existing levels and switches for snapshot monitoring:

Snapshot monitoring levels:

- Database Manager
- Database
  - Application
  - Table
  - Tablespace
  - Bufferpool
  - Dynamic SQL
  - Locks
- Application
  - Locks
  - Statements
- Bufferpool

Snapshot monitoring switches:

- Sort
- Lock
- Table
- Bufferpool
- UOW
- Statement

The status of the snapshot monitor switches can be queried through the *DB2 UDB Command Line Processor (CLP)*, which is a non-GUI, tool or through the *DB2 UDB Control Center,* for example:

```
get monitor switches
```

The output you get looks as follows:

```
Monitor Recording Switches

Buffer Pool Activity Information  (BUFFERPOOL) = OFF
Lock Information                        (LOCK) = OFF
Sorting Information                     (SORT) = OFF
SQL Statement Information           (STATEMENT) = OFF
Table Activity Information              (TABLE) = OFF
Unit of Work Information                 (UOW) = OFF
```

The command syntax for collecting the snapshot information through CLP is as follows:

```
>>-GET SNAPSHOT FOR-----------------------------------------------><
>-----+-+-DATABASE MANAGER-+-------------------------------------+>
| +-DB MANAGER-------+                                          |
| '-DBM--------------'                                          |
+-ALL--+-----+--DATABASES-------------------------------------+
|      '-DCS-'                                                  |
+-ALL--+-----+--APPLICATIONS---------------------------------+
|      '-DCS-'                                                  |
+-ALL BUFFERPOOLS-----------------------------------------------+
+-+-----+--APPLICATION----+-APPLID--appl-id-------+-----------+
| '-DCS-'                 '-AGENTID--appl-handle--'           |
+-FCM FOR ALL NODES---------------------------------------------+
+-LOCKS FOR APPLICATION--+-APPLID--appl-id-------+-----------+
|                       '-AGENTID--appl-handle--'           |
'--+-ALL-----------------------------+---ON--database-alias--'
+-+-----+--+-DATABASE-+-----------+
| '-DCS-'  '-DB-------'           |
+-+-----+--APPLICATIONS-----------+
| '-DCS-'                         |
+-TABLES--------------------------+
+-TABLESPACES---------------------+
+-LOCKS---------------------------+
+-BUFFERPOOLS---------------------+
'-DYNAMIC SQL--+----------------+-'
'-WRITE TO FILE--'
>-------------------------------------------------------------><
```

Based on the combination of levels and switches, your output will contain valuable information for performance and diagnosis purposes.

It is a very common task to monitor the system in order to control the locks that are being held by applications. Suppose you want to know how the locks on your database were influenced by the command `lock table org in`

exclusive mode. The command get snapshot for locks on database sample, when issued from the CLP, will produce the following output:

```
Database Lock Snapshot
Database name                              = SAMPLE
Database path                              =
/home/db2inst1/db2inst1/NODE0000/SQL00001/
Input database alias                       = SAMPLE
Locks held                                 = 4
Applications currently connected           = 1
Agents currently waiting on locks          = 0
Snapshot timestamp                         = 08-24-1999 12:52:59.582179

Application handle                         = 26
Application ID                             = *LOCAL.db2inst1.990824163630
Sequence number                            = 0001
Application name                           = db2bp
Authorization ID                           = ROOT
Application status                         = UOW Waiting
Status change time                         = Not Collected
Application code page                      = 819
Locks held                                 = 4
Total wait time (ms)                       = 0

List Of Locks

Lock Object Name   = 2
Object Type        = Table
Tablespace Name    = USERSPACE1
Table Schema       = ROOT
Table Name         = ORG
Mode               = S
Status             = Granted
Lock Escalation    = NO

Lock Object Name   = 3078
Object Type        = Row
Tablespace Name    = SYSCATSPACE
Table Schema       = SYSIBM
Table Name         = SYSTABLES
Mode               = NS
Status             = Granted
Lock Escalation    = NO

Lock Object Name   = 2
Object Type        = Table
Tablespace Name    = SYSCATSPACE
Table Schema       = SYSIBM
Table Name         = SYSTABLES
```

```
Mode               = IS
Status             = Granted
Lock Escalation    = NO

Lock Object Name   = 0
Object Type        = Internal P Lock
Tablespace Name    =
Table Schema       =
Table Name         =
Mode               = S
Status             = Granted
Lock Escalation    = NO
```

This output indicates that four locks are being held for providing an exclusive lock on table *ORG*. It also shows that tables *ROOT.ORG* and *SYSIBM.SYSTABLES* are being locked in order to provide the necessary lock level for an exclusive request.

The same `get snapshot` command can be issued at any time for displaying the overall locks used on the system.

10.1.1.2 Event Monitor

The Event Monitor is used for recording database related statistics for the following *event types*:

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

More than one Event Monitor can be created, each having a state (1=on or 0=off) that is totally independent from each other.

After an Event Monitor is created, its state can be changed to 1 or 0 in order to start or stop it. When all the information is collected, the event monitor can be stopped by changing its state to 0 in order to analyze what happened during the time the monitor was running.

Event Monitors write their information to file or a pipe. By using the externalized Event Monitor stream formats, users can write applications to extract useful information from the event logs. In addition, there are two

available tools for analyzing the newly created event monitor files: *db2evmon* and the *Event Monitor GUI tool*.

The text-based tool used for analyzing the event monitor files is located in $INSTHOME/sqllib/bin and is called *db2evmon.*

10.1.1.3 Performance Monitor

The Performance Monitor tool monitors database objects, such as instances, databases, tables, tablespaces, and connections. It uses Snapshot Monitor data for viewing, analyzing, and detecting performance problems.

It is a GUI tool that can be started from the *Control Center*, either locally or remotely. It can be configured to send a visible or audible alert, pre-defined messages, or execute commands when certain user-defined threshold values have been exceeded.

These topics are covered in 10.2, “Regular monitoring, ad-hoc, or alert method usage” on page 220.

10.1.1.4 Alert Center

The Alert Center is used by the Performance Monitor to notify the database administrator when the threshold values have been exceeded.

10.1.1.5 Explain

The Explain tool is used by the database administrator to analyze and understand the access plan strategy the optimizer is choosing for retrieving the data as well as the cost of the access strategy in DB2 UDB’s *timerons* measures. The Explain tool can be used to explain the access strategy for both static and dynamic SQL.

The *explain tables*, composed of a set of seven tables, are used by the Explain tool to hold all the information involved in the access plan strategy, such as the command syntax, environment in which the explanation took place, and the operators (FETCH, SORT, TBSCAN) needed to satisfy the SQL. The explain tables are created automatically by the Control Center (when *Visual Explain* is used) or via the command:

```
db2 -tvf $HOME/sqllib/misc/EXPLAIN.DDL
```

EXPLAIN.DDL contains two additional tables for the DB2 UDB Index Advisor, which is a new tool in DB2 UDB V6.1 for recommending indexes for either individual or a workload of queries and/or updates.

The database administrator can choose one of the three available tools for analyzing the contents of the Explain tables:

- Graphical tool
- Text-based tool
- Direct query to the explain tables

Note

It is extremely important that the command `runstats` is executed before using any explain method in order to refresh the catalog tables metadata since the optimizer will read them for creating the access strategy plan.

Graphical tool

The Visual Explain GUI Tool is called from the Control Center panel.

Text-based tools

The text-based tool used for analyzing Static SQL is located in $INSTHOME/sqllib/bin and is called *db2expln.*

The text-based tool used for analyzing Dynamic SQL is located in $INSTHOME/sqllib/bin and is called *dynexpln.*

The text-based tool used for formatting the Explain tables is located in $INSTHOME/sqllib/bin and is called *db2exfmt.*

Suppose you want to use the db2exfmt tool to determine if the optimizer is choosing table scanning when you issue the command `select * from org` against database SAMPLE.

The first step is to set the special register `CURRENT EXPLAIN MODE` to `EXPLAIN`, meaning that only the explain data will be captured and inserted into the explain tables, but the query to be explained will not be executed:

```
db2 set current explain mode explain
```

The second step is to run the SQL command you want to analyze:

```
db2 select * from org
```

The last step is to run the `db2exfmt` command. Accept all the default values by pressing **ENTER**.

```
db2exfmt
```

The generated output describes all the operations, CPU and I/O costs, and table descriptions, such as name, cardinality, and columns retrieved. For this

particular example, the explain output indicates that a table scan was chosen by the Optimizer for data retrieval.

The output you get looks as follows:

```
DB2 Universal Database Version 6, 5622-044 (c) Copyright IBM Corp. 1995,
1999
Licensed Material - Program Property of IBM
IBM DATABASE 2 Explain Table Format Tool

****************** EXPLAIN INSTANCE ********************

DB2_VERSION: 06.01.0
BUILD LEVEL: db2_v6:n990616

SOURCE_NAME: SQLC29A3
SOURCE_SCHEMA: NULLID
EXPLAIN_TIME: 1999-08-26-10.42.00.324351
EXPLAIN_REQUESTER: ROOT

Database Context:
----------------
Parallelism: None
CPU Speed: 1.259585e-06
Comm Speed: 0
Buffer Pool size: 1000
Sort Heap size: 256
Database Heap size: 1200
Lock List size: 100
Maximum Lock List: 10
Average Applications: 1
Locks Available: 1130

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

---------------- STATEMENT 1  SECTION 201 ----------------
QUERYNO: 3
QUERYTAG: CLP
Statement Type: Select
Updatable: No
Deletable: No
Query Degree: 1

Original Statement:
```

```
------------------
select *
from org

Optimized Statement:
--------------------
SELECT Q1.DEPTNUMB AS "DEPTNUMB", Q1.DEPTNAME AS "DEPTNAME", Q1.MANAGER AS
"MANAGER", Q1.DIVISION AS "DIVISION", Q1.LOCATION AS "LOCATION"
FROM ROOT.ORG AS Q1

Access Plan:
-----------
Total Cost: 25.0881
Query Degree:1


      RETURN

      (   1)

        |

      TBSCAN

      (   2)

        |

  TABLE: ROOT

 ORG



1) RETURN: (Return Result)
Cumulative Total Cost: 25.0881
Cumulative CPU Cost: 69979
Cumulative I/O Cost: 1
Cumulative Re-Total Cost: 0.0281971
Cumulative Re-CPU Cost: 22386
Cumulative Re-I/O Cost: 0
Cumulative First Row Cost: 25.0627
Estimated Bufferpool Buffers: 1

Input Streams:
-------------
2) From Operator #2

Estimated number of rows: 12
Number of columns: 5
```

```
Subquery predicate ID: Not Applicable

Column Names:
------------
+LOCATION+DIVISION+MANAGER+DEPTNAME+DEPTNUMB


2) TBSCAN: (Table Scan)
Cumulative Total Cost: 25.0881
Cumulative CPU Cost: 69979
Cumulative I/O Cost: 1
Cumulative Re-Total Cost: 0.0281971
Cumulative Re-CPU Cost: 22386
Cumulative Re-I/O Cost: 0
Cumulative First Row Cost: 25.0621
Estimated Bufferpool Buffers: 1

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

Input Streams:
-------------
1) From Object ROOT.ORG

Estimated number of rows: 12
Number of columns: 6
Subquery predicate ID: Not Applicable

Column Names:
------------
+$RID$+LOCATION+DIVISION+MANAGER+DEPTNAME
+DEPTNUMB


Output Streams:
--------------
2) To Operator #1

Estimated number of rows: 12
```

```
Number of columns: 5
Subquery predicate ID: Not Applicable

Column Names:
------------
+LOCATION+DIVISION+MANAGER+DEPTNAME+DEPTNUMB


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

Schema: ROOT
Name: ORG
Type: Table
Time of creation: 1999-08-23-16.27.04.301208
Last statistics update: 1999-08-26-10.41.45.448796
Number of columns: 5
Number of rows: 12
Width of rows: 47
Number of buffer pool pages: 1
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
```

The explain output clearly mentions, on the second section, that the optimizer chose the *tablescan* operation for retrieving the data. Since this is a small table (12 rows), this was the cheapest way for retrieving the data. However, if you have larger tables, and the optimizer is still choosing the tablescan as the access strategy, you should carefully think about defining indexes for the table, always considering the SQL that is most often used.

Direct query to the Explain Tables

The Explain Tables can also be directly queried through SQL SELECT commands. There are always seven tables populated by the explain tool, given the fact that the EXPLAIN MODE has been set to EXPLAIN:

- *Explain_Instance*
- *Explain_Statement*
- *Explain_Operator*
- *Explain_Argument*
- *Explain_Object*
- *Explain_Stream*
- *Explain_Predicate*

10.1.1.6 LIST APPLICATIONS command

The LIST APPLICATIONS CLP command uses snapshot monitoring to display, at instance or database level, the application program name, authorization ID, application handle, application ID, database name, application sequence number, status, status change, and database path information for all connected applications.

The LIST APPLICATIONS command syntax is as follows:

```
>>-LIST APPLICATIONS----+-------------------------------------+->
'-FOR--+-DATABASE-+---database-alias--'
'-DB-------'
>----+-------------+-------------------------------------------><
'-SHOW DETAIL-'
```

The following is an example for the command and its output:

```
db2 list applications for database sample show detail
```

The output you get looks as follows:

```
Auth Id           Application Name           Appl. Handle
------------      --------------------       --------------
DB2INST1           db2bp                       1


Application Id                           Seq#
---------------------------------------  -------
*LOCAL.db2inst1.990920184212               0001


Number of Agents         Coordinating Node Number
--------------------     --------------------------
1                                     0


Coordinator pid/thread
---------------------------
25722


Status                       Status Change Time
-----------------            ----------------------
UOW Waiting                   Not Collected
```

```
DB Name     DB Path
------------    -----------------------------------------------
SAMPLE       /db2sys/db2inst1/NODE0000/SQL00001/
```

10.1.2 Oracle monitoring tools

The Oracle Enterprise Manager (OEM) is a new product from Oracle that is specially designed for monitoring and tuning databases. OEM typically runs on a PC with Windows NT and connects to the database via SQL*Net.

The following products are extensions to the OEM and can be used to provide the database administrator with a better and easier understanding of the resource consumption within the machine.

The *Oracle Diagnostic Pack* consists of:

- Oracle Performance Manager
- Oracle Top Sessions
- Oracle Lock Manager
- Oracle Trace

The *Oracle Tuning Pack* consists of:

- Oracle Tablespace Manager
- Oracle SQL Analyze
- Oracle Expert

10.1.2.1 Oracle Diagnostic Pack

The following is a brief description of each product within the Oracle Diagnostic Pack.

Oracle Performance Manager

The Oracle Performance Manager provides graphical-based, real-time monitoring that allows the database administrator to monitor the contention, database instance, I/O, memory, and system load. All the data displayed can be stored for replay.

Oracle Top Sessions

Oracle Top Sessions is a tool designed for monitoring all the resources being allocated for connected users. It is possible to display the top sessions, sorted by the database administrator's chosen statistics, needed for the analysis.

Oracle Lock Manager
Through the Oracle Lock Manager tool, it is possible to be aware of all the existing locks on the database and which session is blocking the other session's requested resource.

Oracle Trace
The Oracle Trace tool collects data through a methodology based on every occurrence of key events, such as the number of connections to, or disconnections from, a database.

10.1.2.2 Oracle Tuning Pack
The following is a brief description of each product within the Oracle Tuning Pack.

Oracle Tablespace Manager
This tool is designed to collect data about the internal structure of the database's tablespaces. It is also used to reorganize the tablespaces in order to increase the performance by de-fragmenting and coalescing the small data blocks that might exist inside the tablespaces.

Oracle SQL Analyze
The Oracle SQL Analyze tool is basically designed to tune application SQL.

Oracle Expert
Oracle Expert provides automated tuning through the use of a particular methodology based on integrated rules. It implements a tuning methodology based on the following steps: Specification of tuning scope, collection, view and edit data, analysis, review of recommendations, and then implementation.

10.1.2.3 Comments on Oracle Enterprise Manager (OEM)
Time did not allow this redbook team to investigate these tools in detail. As far as we can determine, most of the information available in these tools is also available using the traditional standard tools of Oracle, but the information is presented in a structured way and saves a lot of time by removing the need to type a lot of select statements or run scripts. From our investigations, these products will make a large impact on the productivity of the DBA. However, there is a charge for the OEM and its extensions.

Oracle Expert is an advanced tool that uses many rules to check and advise changes to the database for performance. It can even create scripts to do the changes. It is doubtful that it fully understands the platform on which it is running; so, it tackles performance from a pure database point of view. For example, AIX and will only make changes within Oracle. Therefore, it will not

optimize AIX level items, such as logical volume options, AIX parameters, and cater for availability via mirrors or RAID 5.

As these tools are new, most sites will not have them. They may find it hard to justify the expense of a high powered PC, the cost of the product, and the time required to install and learn the product, but it should help to reduce performance problems.

10.1.2.4 The UTLBSTAT/UTLESTAT monitoring tool

The UTLBSTAT/UTLESTAT monitoring tool should be used when performance data collection over a defined period of time is needed.

The following two SQL scripts are probably the most important tools in the DBA's toolset for analyzing what Oracle is up to as a whole and for deciding which and what to tune in the Oracle parameters. The scripts can be found in the $ORACLE_HOME/rdbms/admin directory on the system.

The `utlbstat.sql` script (note the *b* means *beginning*) is run, and it creates a set of statistics tables. Then, after a period of the database working, which might be a busy period of the day in a production system or a benchmark test, the matching `utlestat.sql` script (note the *e* means *end*) is run. This creates a report.txt file in the current directory. This human-readable file includes:

- The SQL used to collect the data so that you can look into the meaning of the columns in the Oracle documentation.
- Explanations on what to look for in the numbers in the report.
- Some ideas about what can be changed to improve performance.

The statistics compare the *before* and *after* values of a lot of important Oracle counters. So, make sure that the system is doing the sort of work that you wish to gather information about in the period the monitoring tool runs. For example, if you are tuning the daytime performance, do not capture data overnight.

The following is a small sample part of the output generated in report.txt:

```
. . .

SVRMGR> Rem Select Library cache statistics.  The pin hit rate should be
high.
SVRMGR> select namespace library,
2>        gets,
3>        round(decode(gethits,0,1,gethits)/decode(gets,0,1,gets),3)
4>           gethitratio,
5>        pins,
```

```
6>        round(decode(pinhits,0,1,pinhits)/decode(pins,0,1,pins),3)
7>           pinhitratio,
8>        reloads, invalidations
9>   from stats$lib;
LIBRARY            GETS GETHITRATI      PINS PINHITRATI  RELOADS INVALIDATI
---------------------------------------------------------------------------
BODY                  0         1         0         1         0          0
CLUSTER               0         1         0         1         0          0
INDEX                 0         1         0         1         0          0
OBJECT                0         1         0         1         0          0
PIPE                  0         1         0         1         0          0
SQL AREA            114      .956       263      .947         4          0
TABLE/PROCED         26      .885        40        .9         0          0
TRIGGER               0         1         0         1         0          0
8 rows selected

. . .
```

This small sample outputs a value of .956 for `SQL AREA GETHITRATIO`, which is an ideal value since it represents that 95.6 percent of the parse calls could find a cursor to share. This value should always be in the high nineties.

It is worth running these scripts and carefully studying the output and recommendations found in the report. The report includes the following sections:

- Library cache statistics
- 96 important statistics covering the measuring period
- System-wide wait event
- Latch statistics
- Buffer busy and wait statistics
- Statistics for roll back segments
- init.ora parameters currently in effect
- Sum I/O operations over tablespaces
- I/O spread across drives
- The times that `utlbstat` and `utlestat` were run

All the above areas will display valuable performance information. Choosing which area to monitor first will depend upon which are the bottlenecks on the database.

10.1.2.5 EXPLAIN PLAN command

The Oracle `EXPLAIN PLAN` gives the DBA, developer, or anyone who is writing SQL statements the means to:

- Find out which method the Oracle optimizer will use to access the data
- The order in which the tables are extracted from the database
- If indexes are used
- The relative cost of the statement

No book about Oracle is without a chapter dedicated to explaining the `EXPLAIN PLAN` mechanism and how to read the output. The original, but basic, method, using sqlplus, is still available, but is hard work. This involves:

- Creation of a special table
- Running the `EXPLAIN PLAN` command
- Using a `SELECT` from the special table to extract the plan

The new Oracle tools for PC based users (see below) make the `EXPLAIN PLAN` far easier to use. Note that the database does not actually run the SQL. It only gets the RDBMS to analyze the SQL and the optimizer to decide the plan it would use to run the query.

Which ever method is used, the output of `EXPLAIN PLAN` is useful to investigate what Oracle is doing with particular SQL statements. This is very useful when:

- Oracle seems to take unexpectedly long to run a statement.
- Oracle seems to take an unexpected method to answer a query.
- To investigate the effects of parallelizing a query.
- Alternative SQL statements might yield a result faster - This is extremely useful for analyzing decision support statements that might differ in their execution time in terms of hours.

Prior to running the command, the Explain tables should be created using the `utlxplan.sql` script.

Once the Explain tables are created, the command `EXPLAIN PLAN FOR` can be issued prior to any SQL statement.

Suppose you want to analyze the access plan generated for the query:

```
SELECT * FROM EMP
```

The first step is issuing the following command:

```
explain plan for select * from emp
```

The command will be executed, and the explain data will be stored in a table called *PLAN_TABLE*. To view the execution plan contained in the PLAN_TABLE, you should run the following SQL statement:

```
select id, operation, options, object_name, position from plan_table
```

The output you get looks as follows:

```
ID OPERATION        OPTIONS OBJECT_NAME POSITION
-- ---------------- ------- ---------- --------
0 SELECT STATEMENT
1 TABLE ACCESS       FULL   EMP          1
2 rows selected.
```

The value `FULL` for *TABLE ACCESS* entry explains that the optimizer chose tablescanning for all the tables in order to retrieve the data.

For more detail on the use of `EXPLAIN PLAN` with Oracle, refer to the Oracle documentation. For example, the *Oracle 8 Server Tuning manual*, A54638-01, has a complete chapter on the use of `EXPLAIN PLAN`, and the *Oracle 8 Server Concepts*, A54646-01, has a chapter on the optimizer, which includes explaining the `EXPLAIN PLAN` output.

For a nested output of the `EXPLAIN PLAN`, which can help in complex SQL statements, see Appendix B.2.7, “Oracle nested explain plan” on page 375.

There are also excellent reference books on Oracle that include this subject. See the references section at the end of this redbook.

10.2 Regular monitoring, ad-hoc, or alert method usage

Although we can propose three different methods of monitoring, regular, ad-hoc, or alert, it is important to highlight that the best choice will vary from administrator to administrator.

The monitoring task is very important in order to:

- Maintain a minimum level of control over the system environment - The more the establishment of the monitoring process is delayed, the more complex it will become to put the environment (HW and SW) under control.

- Prevent the machine and operating system from affecting the database, for instance, through external factors, such as slow disks and CPUs, lack of memory, or older versions of operating systems.
- Preventing the database from affecting the machine and operating system - If the database is consuming more memory, disks space, and CPU than it is supposed to, the operational system can be severely impacted.

Each monitoring session consists of basically three steps:

Define your objectives
You have to define exactly which database or AIX area you want to monitor. For example, you might want to know how your queries are affecting AIX memory consumption or how many disk I/Os your applications have done so far for retrieving the data.

Define information you want to analyze
Once you know which area you want to analyze, you have to figure out the possible ways to achieve the results. For example, to find out how the queries are affecting AIX memory consumption, you could start the analysis by taking a look on how much memory space the database is allocating for the sort operation and also check on AIX how memory and paging space are consumed when the applications are running.

Define which tools will be used
Once you have chosen what to monitor and how to monitor, you have to choose which tool will be used. You may use the pre-defined monitors, or you can also create your own monitoring scripts.

All three steps must be followed every time you begin a monitoring session.

10.2.1 Regular monitoring method

This monitoring technique is an essential daily routine for both system and database administrators.

The main purpose of this close and regular monitoring is to:

- Keep the AIX and database under supervised control
- Document how the system and the database are performing day-to-day
- Based on the collected history, try to narrow down the possible predictable incidents

The system administrator and the database administrator involved with the regular monitoring should have the following prerequisites:

- Global system and database knowledge
- Total knowledge on their expertise area
- Ability to manipulate the monitoring tools
- Ability to create their own monitoring scripts

They should also be committed to allocate, every single day, a time period for this monitoring process.

10.2.2 Ad-hoc monitoring method

Although the regular monitoring method guarantees that the system is, most of the time, under total and assisted control, some unpredictable facts, such as unexpected delays or hangs, might sometimes happen.

For this specific scenario, the practice of regular monitoring will assist the database administrator in figuring out the main cause of basic problems, such as if an application is really hanging or just performing long sorts, if an application is stopped due to a lock or deadlock situation, or if a query response time is too low due to a disk I/O bottleneck. A good practice is to have some pre-defined scripts already created that can speed up the problem determination by displaying basic system characteristics, such as CPU consumption, disk usage, paging space consumption, number of users connected, and number of AIX processes allocated to the database.

Please refer to Appendix B, “Vital SQL” on page 371 for more information about the suggested scripts.

10.2.3 Alert monitoring method

The alert method consists of pre-defining some lower and upper values for special performance variables, and, when these values are reached, the database or system administrators are notified, and then an action can be taken for solving that possible problem. This alert method can also be used with the regular monitoring method but, due to its unassisted characteristic, should not be the only monitoring technique adopted for an environment.

10.3 Performance monitoring scripts

Once you are familiar with all the different monitoring tools and are aware of all the possible monitoring strategies, you will be able to determine which tool best fits for your daily database’s monitoring needs and also create some scripts that, using the output from the monitoring commands, can help you speed up the monitoring process.

It is also recommended that you store all the daily database monitoring information so that you can have a better and cumulative understanding of how the machine resources are being consumed by your database. This monitoring history could help in determining how the database is growing and also help to plan for more memory and disk resources before these issues become bottlenecks.

Please refer to Appendix B, “Vital SQL” on page 371 for more information about the suggested scripts.

10.4 Monitoring and tuning responsibilities

This is one of the most discussed topics when performance is an issue.

A poorly performing database does not necessarily mean that the hardware does not meet the system demand, or that the database design was poorly planned, or that the application programs were coded badly. However, one (or all) of them could be the cause of the database’s bad performance.

Performance in focus must be a reality for everyone involved with RDBMSs, from application developers and system designers through the database administrator. A well-tuned system takes into account not only the database itself, but all the areas that, in conjunction with the database, form the system.

Prior to designing the database structure, there is a need for understanding all the database’s resource demands regarding disks, memory, and processors. If the hardware was not very well planed for meeting all the user’s demands, this is the first possible area to be considered.

We can basically point out three different job responsibilities that are in charge of almost all possible enhancements in the RDBMS performance area:

- The system administrator
- The database administrator
- The application developers

The system administrator should always be aware of how the databases are affecting the physical structure and also provide a stable and secure operational environment for them.

The database administrator’s main tasks are: Design, develop, operate, safeguard, and maintain one or more databases. They can also be responsible for the database integrity and availability by managing the

authorities and privileges over all the objects as well as controlling the backups.

The application developer is responsible for developing and testing applications that issue SQL statements for inserting, deleting, updating, and retrieving data. This is a special area where a poorly coded SQL statement is able to influence the behavior of the entire database. You can influence the RS/6000 memory and swap disk consumption depending on the amount of resources the database will have to allocate for the SQL statements to be executed. Please refer to 2.6, “Structured Query Language” on page 29 for more details about using SQL Statements.

It is important to say that the three areas are inter-dependent, and that a change in one of the areas could possibly affect the other ones.

In almost all the cases where performance problems show up, the database administrator is usually the person capable of pointing out where the main performance bottleneck is.

Note

The monitoring process is a team task and will only be successfully performed when all the involved areas agree and work together targeting a stable and controlled system.

10.5 When should a performance problem be reported and to whom?

Performance problems, in a monitored and stable environment, can be detected by the administrators but are usually reported to them by the end user.

After a problem is reported, problem determination monitoring should be done involving the database administrator, the system administrator, and the application programmer.

10.5.1 What are you looking for?

CPU

When the CPU is busy 90 percent of the time, it is considered that it has reached its capacity. It is also a good practice to monitor how the CPU is being consumed by AIX. The operating system should not consume more than approximately 20-40 percent of CPU time. If this value is higher, check with performance tools, such as trace or tprof, to find out which application is causing the high system CPU time consumption and continue your

investigation from there, for instance, find out if the application can be changed so that it consumes less system CPU time.

In multi-processor systems, the system administrator should monitor if the CPU load is balanced across all of the CPUs.

Disk
The database administrator should be aware of the fact that wrong physical data placement could lead to data skew and, sometimes, overload the I/O requests for some disks while others could be available and not processing.

The I/O requests should be equally distributed among the disk controllers and disks.

Memory
The Oracle RDBMS uses memory for manipulating the SGA, background and user processes while DB2 UDB uses it for the bufferpools, database heaps, internal processes, and agents.

The more memory you can allocate for the database, the higher the *buffer pool hit ratio* tends to be. The buffer pool hit ratio indicates the percentage of time that the database manager did not need to load a page from disk since it was already available in memory. The greater the buffer pool hit ratio, the lower the frequency of disk I/O.

Paging space
The system and database administrators should be aware that, whenever you use the operating system paging space, the performance automatically decreases since the number of disk I/Os and CPU consumption increases.

The amount of disk space defined for the AIX paging space will depend on how the database allocates the real memory resources. If the amount of real memory is enough to supply the RDBMS with all the memory requested, the paging space can be set to the default value recommended by the AIX *Installation Assistant* tool.

Sort operation
The application developers should avoid using SQL statements that can generate sort operations, such as `ORDER BY` and `GROUP BY` clauses.

If the sort operation cannot be avoided, it is recommended that you allocate enough room for this operation to be done in real memory and, thus, avoid using the paging space.
DB2 UDB allows you to control the allocation of the sort heap by using the sortheap and sheapthres parameters (see also 12.5.5, “Sort heap size

(sortheap)” on page 276 and 12.5.6, “Sort heap threshold (sheapthres)” on page 277).

Locks
The database administrator should always control the number of locks on the database. Both RDBMSs lock data using the lowest level of restrictiveness, the row level, allowing the highest degree of concurrence while guaranteeing data consistency.

The application developers should keep in mind that the frequency of `COMMIT` statements usually determines the degree of concurrence. The more `COMMIT`s the application developers code, the less rows will be locked, thus, allowing other users to use the rows. Although each commit operation increases the number of physical I/Os to disk in order to record the new row value, it is still recommended that, whenever suitable, `COMMIT` statements are used.

For information on DB2 UDB lock parameters, see 12.5.13, “Maximum storage for lock list (locklist)” on page 283 and 12.5.14, “Maximum percent of lock list before escalation (maxlocks)” on page 284.

Deadlocks
A deadlock situation happens when two or more applications are waiting indefinitely for data locked by each other.

Although this situation is automatically solved by both RDBMSs and does not need external intervention, a high number of occurrences can point to possible lock contention, and the application’s code should probably be reviewed as well as the isolation level.

Chapter 11. Tuning an RDBMS system

Everyone wants value for their money, and the point of tuning an RDBMS is to make sure that the system is delivering good performance. In tuning an RDBMS, there are two approaches:

1. Minimum man-power approach

   Setting up a system that provides reasonable performance with only the minimum amount of man-power used for the on-going tuning effort of the System Administrator (SA) or Database Administrator (DBA). In this case, once set up, the machine is largely ignored unless users complain or something goes wrong.

2. Maximum performance approach

   Setting up the system for maximum performance. Tuning of the system includes the system administrator monitoring the hardware and AIX, and the DBA monitoring the database and applications on a daily basis. In this case, the machines are likely to be larger and, thus, of higher value. The investment in man-power is justified in efficient use of the computing resources.

It is important to know which you are trying to achieve, and it depends on the company culture and the importance of the system itself. This decides the time invested in tuning and the investment level in extra capacity.

Regardless of which approach is used, the RDBMS will be tuned due to one of five causes:

- Regular task

  Regular periodic monitoring and tuning is standard practise. Many sites do a review of performance on quarterly, half-yearly, or yearly intervals. Problem machines would be investigated immediately.

- Generated warning

  The automated monitoring of the system has warned that performance is degrading and has hit some threshold. Please see Chapter 10, “Monitoring an RDBMS system for performance” on page 203 for more on these subjects.

- Emergency

  There is an emergency in performance or response time, which has been highlighted by user feedback. The tuning must identify the problem, recommend a solution, and then work out how to avoid this happening again.

- New system

  A newly build system requires initial tuning for maximum performance before going into production. In most of the cases, however, this system might already be in production because of the difficultly of generating user workload artificially and not being able to predict real user workloads and work patterns. Ideally, the number of users is still growing and is not the full number yet since this will allow tuning before system response times become unacceptable.

- System change

  The system is shortly going to have a change in workload. For example, the database size increased, the number of users increase, or a whole database is added to the system for concurrent access. In this case, the current system needs to be tuned to free up as many resources as possible before the new workload is added.

Whatever the reason, the approach will largely be the same, although, in the case of an emergency, there is much higher pressure from users to get to the root of the problem and fix it.

11.1 Tuning skills

Most RDBMSs are important. They represent an important investment by the company. Any RDBMS is complex. They include state-of-the-art architecture machines, such as the RS/6000 family; they have a complex operating system, such as AIX, which offers many options for tuning and particularly disk layout, and then advanced databases - again with many tuning and performance options. On top of this platform is a complex database data structure and application software.

This means an RDBMS is both important and vital. The tuner, if not careful, could make performance worse, or even damage the database, and make it unavailable for a long period of time. To reduce the risk, make sure that the appropriate skills and knowledge are available before tuning a system. These might include:

- RS/6000 and AIX architecture
- AIX System Administration
- AIX tuning and performance
- DBA skills for the RDBMS
- Tuning for the RDBMS

- SQL tuning
- Application writing

If you are unsure either:

- do not tune
- investigate and propose changes to the system for others to check before implementing the changes
- start building a team that together has all the right skills

Note

With RDBMS tuning, a little knowledge is a very dangerous thing.

11.2 Reference manuals and books

Before you start tuning, make sure that you have the right reference materials readily available. While tuning, many questions will be raised, and these need to be answered quickly and completely.

For AIX, there are two excellent and highly recommended redbooks covering performance, sizing, and the tools:

- *Understanding IBM RS/6000 Performance and Sizing*, SG24-4810

  This has full explanations of how the RS/6000 really performs and why. This is a good book to expel years of misconceptions, muddy thinking, and false ideas that are found in the computer industry.

- *AIX Performance Tools in Focus*, SG24-4989

  This is the best place of information on the wide range of tools available for AIX, and you will need to understand many of these tools for database tuning. UNIX commands and tools are famous for having high-detail levels but very poor headings and explanations. This book actually explains what the numbers mean and what you can do to change them.

For the RDBMS, there are the manuals for the database. In particular:

- The performance tuning manuals for your actual RDBMS in the correct version. These may be online versions due to costs, however, hardcopies of the tuning manuals might be better for scanning and finding the right section.
- The RDBMS reference manuals for the RDBMS and DBA tools.

- The RDBMS SQL reference manual.
- The RDBMS introduction or concepts manuals.

There is a growing number of good performance books available. For a few recommended RDBMS tuning books, see Appendix F.3, “Other resources” on page 415.

11.2.1 About RDBMS tuning and RDBMS performance tuning books

If you purchase an RDBMS Performance Tuning book, or read the Tuning chapters of a general book on your database, one thing you will notice is that there is no end to the details that can be covered in these books, and each generation of the book seems to be larger than the previous. Next, you will find that there are some guidelines about where to tune and where performance stems from. This highlights an important message about what can be fixed in performance terms on the actual machine. They will quote some numbers, such as:

- 50 percent of the performance comes from a good database design and application design.
- 30 percent of performance comes from good programming and correct use of SQL.
- 19 percent comes from good implementation and tuning the system.
- 1 percent comes from fixing hardware errors.

The numbers might vary a bit depending on the source. This really means that if the design and programming is bad, or even just poor, then, in practical terms, there is little that can be done on the system that will remedy this. If these parts of the total performance picture are poor, the machine could run 1000 times slower than expected. If, however, these are good, then the tuning on the machine can be the cause of performance problems. A very poor implementation can cause a 100 times slow-down in performance.

While this is very interesting, there is little or nothing we, as system administrators and database administrators, can do about the design, the programming, or the SQL. The design was fixed months or years ago. If the application was bought as a package, then the application code is not even available. We cannot get the code changed, or it means a very large expense. If the code is developed in-house, then we might find that the development team is too busy on other projects. We might be able to make the case for specific SQL changes, but only if we can identify a very small number of problem statements.

For the most part, we can only alter the set up and configuration, and if we can make the business case to justify the expense, purchase a little extra hardware.

In this redbook, we have assumed there is little we can do about the SQL being sent to the RDBMS. We just have to make it work as fast as possible. This means that performance tuning books that have 70 percent of the book taken up with SQL tuning tips are not going to help the DBA that much. We have to concentrate on the AIX and RDBMS side of tuning.

Note

Remember performance is only *refined* by SA and DBA.
It is *created* by the designers and programmers.

11.3 Tuning strategy

Before we get to the tuning hints and tips, we need to decide the tuning method to use so that the tuning exercise:

- Does not waste time - especially if we are tuning due to an emergency because users cannot perform their tasks.
- Does not waste computer time and people resources.
- Actually comes to a conclusion.
- Has some quantitative measure of the improvement made.
- Can be applied elsewhere, if appropriate.

Many books and papers on performance tuning detail the formal tuning method and include chapters on:

- The iterative process
- Only changing one thing at a time
- Defining the goals and objectives
- Reproducible workloads
- Careful instrumentation, measurement, and documentation

All of the above list is really common sense, but:

- It can be hard to follow in practice.
- It would mean tuning would take a very long time.
- There are fundamental things that need to be addressed immediately.

So, in addition to this normal fine tuning method, there is the *change all at once* approach to tuning that has the following phases:

- Gather all the information about what is actually going on and disregard the rumors, opinions, and theories about the problem.
- Fix the blatantly obvious mistakes in one pass.
- Get the system into reasonably good shape by tuning the high-impact performance options including hardware, AIX, and basic database parameters.
- Then, start the fine tuning phase using the formal tuning method.

Database Tuning

It is much easier to tune an RDBMS badly than tune an RDBMS well.

The following sections deal with formal tuning and the change all at once approach. They are followed with our list of tuning hints and tips for particular databases.

11.4 Formal fine tuning method

There are many books on this subject; so, this section is a summary of the important points and ideas about the formal and fine tuning method. Most will seem obvious and common sense, but they are here to stop one classic mistake, which might be called the *all guns firing approach*. In this approach, every option and parameter is changed, seemingly at random, and no one knows if the performance got better or worse. This has three possible outcomes:

1. A miracle happens, and the performance becomes excellent.
2. Performance decreases dramatically, and nobody knows which parameter was the cause.
3. Tuning goes on forever, and, eventually, a manager stops the tuning as a waste of effort and money and appoints someone with a real method.

The formal tuning method can be outlined based on the following principles, outlined in 11.4.1 through 11.4.14.

11.4.1 Clear definition of the success criteria

Before actually doing anything on the system, the performance tuning team needs to define the goals and objectives. This is obvious but rarely done at the start.

Often the machine has performance problems, and the goal is to remove them. This is not a good goal because it has not definitive end point. Try to answer the questions: How can you determine if the system performance is good enough? How is the performance actually measured?

Another poor example is: *The response times must be acceptable*. Acceptable to whom, and how is it decided if it is good or bad? It might depend on how the system administrator is feeling that day or how many complaints from users come in, and might not be based on the system at all. A further poor example is: *The response times must be less than 3 seconds,* but often the application has some tasks that are expected to take 10 minutes, such as creating a large report. The response time requirements need to be limited to particular transactions and, hopefully, the ones most often used.

Again, you need clearly defined and measurable goals.

You might be tuning the online performance, but the problem is with the batch run. For late night batch runs, there is often a goal set that it must take less than a certain number of hours. But do not forget there might be an alternative solution, such as performing an online backup rather than an off-line backup and, thus, increasing the time available for the batch run.

If you are given a number of targets to reach, then insist they are given a priority order since it will help you to make better progress. For example, when you have fixed four out of five problems, you can claim to have fixed 80 percent of the problem, and it is the least important one that remains to be finished.

11.4.2 Limiting the activity

Performance tuning is a never-ending task. The data, users, and workload changes with time and more tuning can be performed. So, unless you have been told to tune this system forever, at some point of tuning, you will have to stop. A decision on the time frame allowed must be made. This might be a limited number of days or a limit to the performance gains reached. If the limit is time based, then the team should aim to increase the maximum amount of performance in that fixed period of time to increase user satisfaction. If the tuning is performance goal based, then the team should focus only on that

goal and to achieve it in the minimum amount of time and effort. Most of the time there are actually both limits, but one is going to be a lot harder to reach than the other. So:

- If time is limited, work on the quick wins that can be tried in the time.
- If the goal is limited, work on all the options in impact priority order that can help reach the target.

11.4.3 Iteration

Tuning is an iterative process. A test is run, the results are studied, a change is made, a further test is run, and so on. This is obvious but has large implications. First, you have to define what is a test. This can be very hard to be precise on (see 11.4.8, “Reproducible workloads” on page 236) and you have to be careful to collect the right results. Next, the changes made have to be carefully controlled (see 11.4.4, “One change at a time” on page 234). Continue to iterate until either:

- The original goals are met.
- You run out of time.
- You run out of ideas and areas to tune.
- You proved it cannot be done and need the goal post moved.

Note

Iterate and then iterate again!

11.4.4 One change at a time

Only change one thing at a time between test runs. If, as the result of a test, you want to try to change two variables, the temptation is to change them both and rerun the test. But, when the results show a small improvement, you cannot determine what caused the small improvement. One change might have made all the difference. The other change might have made no difference; it might have stopped the first from making an enormous improvement, or it might have reduced the performance, which is hidden by the improvement of the other. The only reliable method is change one thing at a time. This is obvious but it is very tempting to make a lot of changes (see 11.5, “Change all at once method” on page 241).

In practice, all tuners change multiple things at a time and are forced to by time limitations, but they try to limit the function area. For example, tune just the allocation of memory to the various memory consumers but do not tune the disk layout at the same time.

11.4.5 Deciding priorities

Once a test is run, and the results are analyzed, there will be a number of conclusions. For example, the buffer cache size, number of locks, and perhaps the AIX parameter to free up memory should be tuned. The team then must decide which area to tune next.

There needs to be a balance between how hard it is to tune this area, the benefits in performance, and, if the machine is in production, the likelihood that this could cause a major problem if it goes wrong. A list of the priorities needs to be drawn up. Usually, the safest and most effective changes are the route to go. Sometimes good choices are ignored because the team does not have strong skills in this area.

Some effective teams draw up on a white board all the alternatives. They then discuss each option and try to estimate the potential performance improvement. Finally, they all vote on which option to try in the next round of tests.

11.4.6 Hot spots

When investigating the machine's performance, one of the CPU, memory, disk, and network areas will become the focus of your attention. This area seems to be the bottleneck area, and it needs addressing. It is called the *hot spot*. But be aware that, if this area is fixed, then the hot spot will just move elsewhere. For example, we can reduce data disk bottlenecks by increasing caching, but that might cause high paging rates. We need to tune the system to make it balanced. We can express this as having a lot of small bottlenecks all over the system, somewhat impacting performance, or that no one area is the hot spot, but all areas are getting warm.

There is always one hot spot in a computer system. After all, in a perfectly tuned system, there is always something stopping it from reaching infinite performance and zero response times. The point is, if all areas of the machine are well used, the apparent hot stop must be regarded as normal.

Also, note that the hot spot can move around. As users do different tasks during the day, week, or month, the hot spot might change in the system from the lack of CPU in periods of high OLTP transaction rates to the lack of disk speed during report creation at a different time. Also, the needs of the nightly batch work or data loads and summaries can be very different to the needs of online users during the day. Many sites have different tuning setups for daytime and nighttime activities.

Note

Hot spots are the places to focus on.

11.4.7 Well known important areas

There are hundreds of options and parameters that can effect performance in a modern computer system. These range from hardware choices and connections to AIX and on to the RDBMS parameters and configuration. But there is a list of well known things that have the largest impact on performance. Also, check the list of well known simple mistakes (11.8, “Classic mistake list” on page 257) to make sure you avoid them.

11.4.8 Reproducible workloads

This is a particular problem with systems that have many online users. The users’ work patterns change, even during the day. They may do different things in the morning and afternoon. For example, mornings might be telephone sales, and the afternoon mail order, and later in the day, reports. Most businesses have peaks in orders on particular days of the week or times of the month and year. Also, every business has business administration induced peaks, such as end of quarter and end of year. To make matters worse, users also can work at different rates at different times. This causes a problem when the workload changes between tests. It can be very difficult to determine if the performance difference is the result of the tuning effort or the users changing their work pattern.

One last problem is that if performance improves, and, therefore, the system has better response times, the users simply do more work, and the system slows down again. This means the transaction rate is higher, but the response time remains unchanged.

What we need to do is to capture and compare both, the response times and the transaction rate, before determining if the performance is better or worse.

Batch workload, in comparison, is quite sane, but do check the numbers in this case also. For example, you might find there are differing numbers of records used in batch runs on different days of the month.

Decision Support Systems have an even worse problem in this area. The transactions are usually 10 to 100 times (or much more) bigger than the transactions of an OLTP system. This means that different queries can differ in response times by large amounts; so, measuring the response time and throughput is still important. The extra problem is that the size of the SQL

statements can vary a great deal when compared to other systems. DSS queries can differ in complexity by 1 to 1,000,000 simply by changing the criteria for the SQL statement. For example, looking for a trend of a particular product, a small time period, and a limited number of stores might take 15 seconds; but looking for all products, a large time period, and all stores might take 30 hours, simply due to the volume of data and that summary tables cannot be used. Also, many sites bring new data onto the system irregularly, sometimes just once a month. In the next few days, this new data is investigated intensively, and, therefore, the queries that are submitted change during the months. The only suggestion to help this is to also measure large table scan rates and the volume of data extracted from the disks, or to develop a standard and typical set of queries for testing.

11.4.9 How to measure response time

This is a problem for online user systems because it is very difficult to accurately record the response time from a system. If the system is extremely slow, then there will be no discussion that the response time is not sufficient. But if the system is responding in about 1.9 to 2.1 seconds, and the requirement is 2 seconds, there can be a large argument about how it is measured. A human cannot start and stop a stop-watch with an accuracy of 1/10th of a second anyway. Also, note that a fixed response time goal (for example, all response times must be under 2 seconds) is unlikely to be achieved nor guaranteed for two reasons. First, there are peaks in user demands that are unpredictable, for example, if all the users attempt to commit a transaction at the same time (this is rare but can happen). To guarantee the response time for this extreme case would mean a system at least ten times more powerful than really warranted. Second, there are always transactions that take too long. For example, creating a report or certain wildcard lookups of large numbers of rows (such as looking up a customer with only part of their name) that will take longer than the 2 seconds and which no one really would expect within this time limit. This results in a more realistic response time goal, for example, 90 percent of responses are under a fixed time limit and for specific transactions (usually the most used 10 or 20 transactions). Again, this makes statistically accurate measurements extremely hard to agree on.

Note

Trusting the users to give a accurate picture of response times is famous for being inaccurate.

One approach in the industry is that the application code is modified to keep track of the response time by measuring and reporting the response time.

The down side is that this takes CPU power, and the data needs to be collected and summarized. This extra work results in slowing the system down. But, we may see more of this in the future.

The alternative is running a small tool that, while the users are working, executes some well know SQL statements that are as typical as possible to the user workload, and the response times of these are accurately measured. This sample workload is small compared to the user workload and is run at regular intervals to allow the performance tuning team to accurately determine the relative response time of the sample. These accurate response times help in working out the benefit (or not) of tuning changes.

11.4.10 Careful instrumentation and measurement

Tuning is impossible unless you have the means to decide if you have made an improvement or not. This requires you to take measurements on the system to help decide what the problem is, and later, to decide if performance has improved due to tuning changes. Instrumentation is an IBM term that means the system has the right tools and features to allow the data to be collected, be meaningful, precise, and not intrusive. This instrumentation should not impact performance itself more than a few percent. Fortunately, AIX is extremely well instrumented and has powerful tools beyond the standard UNIX system. This includes AIX tools, such as Performance Toolbox/6000 and the AIX trace system. The DB2 UDB and Oracle databases have excellent tools as well. In all cases, they need to be used and understood before this benefit can be harnessed for making performance enhancements to the system.

11.4.11 Documentation

Although some dread this word, it does not need to be painful in the case of performance tuning. The bulk of the facts needed for tuning can be collected from the tools. You do need to be careful that we save these results in a methodical way. In addition, you need to make notes of what was done in each test in terms of when the tests were run, the particulars of the workload and users, the interesting facts discovered, and suggestions for improvements. This sounds like a lot, but it is worth to formally record this data. This can either be in a tuning log book or in read.me type files saved along with the tool output files and results. If this is neglected, and you later want to refer back to earlier tests, you will find that you have forgotten or are unsure of the facts. This then means rerunning the test to be sure. So, we recommend keeping good records of each test, as this will save time. It also helps writing these things down because you have to think about how sure you really are of the facts and how much is guessed.

Tuning log recommendations:

- First of all: Create a log. Without a clear and detailed log, it is very easy to forget the configuration details, the conclusions of each test, and the set of results. This results in having to rerun a test in order to check it, which wastes time.
- Perhaps include parts of the HW and DB logs of the system.
- We recommend to use a lot of directories and use a standard script to capture all the configuration details and then add a read.me about what the test hoped to prove and what it did prove.
- Collect data with scripts, such as those in Appendix B, “Vital SQL” on page 371 and with database supplied tools, such as `utlbstat.sql/utlestat.sql` for Oracle or `snapshot` for DB2 UDB.

11.4.12 Scheduling the tests

If the system is not in production, then you have an excellent chance of making quick progress, and you have time to try out some ideas. It is good to try to reduce the test period to a time as short as possible. 20 to 30 minutes is a good time. Keep in mind that large systems take a while to reach a steady pace, for example, to get all the users working and the RDBMS buffer cache or pool filled with data. If the test is longer than 20 to 30 minutes, this drastically reduces the number of test cases that can be run in one day.

If the system is in production, then there are a number of problems to overcome. First, is that if you make a mistake in tuning the parameters, then all the users will be complaining the next day. Second, many of the tuning options require the database to be restarted. This is not an option on production systems, and many international companies might only allow this once a week. This cuts down the number of opportunities you have to improve performance. In this case, a test machine is the only real option, but, ideally, this needs to be a similar size to the production machine but this can be hard to justify because of the costs. Alternatives are renting machines or using IBM test centers to run a tuning project or having sole access to the machine, but this usually means early hours, such as three to five in the morning or on Sundays.

11.4.13 Verifying the improvement

The change was made and the test rerun. Did it make a difference? There are a number of possible outcomes:

- Big improvement - Good, a further change might be in order to see if a larger change would cause a larger improvement.
- Small improvement - Fine, you might try again or change something else that might help even more.
- Tiny improvement or degradation - You have to decide if this was worth it or not. You might find that the margin of error means that you cannot tell if anything improved or not. You have a choice of leaving it at the new level or returning it to the original.
- The before and after performance was not checked - Not very professional, and might well cause a bottleneck later.

We recommend:

- If the old level was a default, then return it to the default.
- If you think the new number should really help but suspect there is another item that stopped this improving performance, then leave it at the new level and investigate the other item further.
- If this change was to remove the current bottleneck but made little difference, then return it to the original value and think again.

11.4.14 The tuning team

Do not forget that you are not alone. There is help available from your supplier, as it is in their best interests that the hardware, software, and RDBMS are working well and meeting your needs because they know happy customers are likely to buy more in the future. IBM, IBM Business Partners, application providers, and the database vendors have support organizations that can assist you if you cannot solve the performance issues.

When you escalate to other people, they have to start from scratch; so, save them time (and possibly your company money) by getting them the facts and information before engaging with them. Then, be ready to get further information that they might need and be ready to run other tests and, if necessary, add instrumentation so that they can investigate at a higher level of detail. Do not expect to phone in a *it does not work* problem and expect the support people to solve it from there.

See Chapter 14, “Austin - we have a problem!” on page 345 for more information on getting assistance.

11.5 Change all at once method

In the previous section, we have outlined a good approach to tune a system for maximum performance and minimum response times. If the system is working reasonably well, then this is an excellent approach to fine tune the system further.

But, often the system performance is very poor, or this is the first ever tuning session on the new system. In this case, you might use the alternative *change all at once approach* to try and get the system working reasonably well and as quickly as possible.

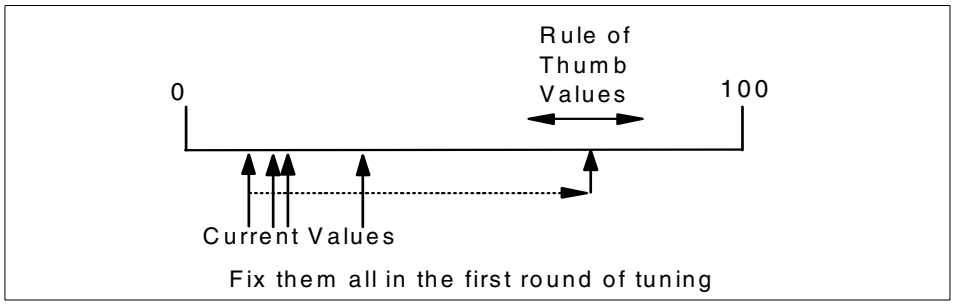


Figure 30. Change all at once method puts all the standard tuning parameters right in one go

This approach involves six key stages:

1. Ignoring the rumors about what is wrong, as these are based on little or no information.
2. Check for errors.
3. Get to the latest fix level.
4. Measure what is actually going on regarding performance and then document all the parameter settings.
5. Check and change the top ten performance parameters to sensible settings and use defaults for most of the rest.
6. Remeasure performance to check if performance has improved.

The point of using a change all at once approach is to get the system into good shape as soon as possible. Most of the tuning parameters should be the default values apart from a few, well known, key ones that need to be set depending on the size of the machine and the workload type. You also need to check that the machine is configured properly so that it can make good use of the various parts available.

Once this is done, you need to remeasure the system performance. Hopefully, it has improved. Then you can do further tuning using the formal tuning method.

The above stages are covered in more details in the following section.

11.5.1 Ignore the rumors

The IBM technical support groups often get involved with performance situations that are escalated through customer management and then through IBM. By the time a technical specialist is involved, there have been gross distortions in what is the actual problem and what are the symptoms leading people to decide what the problem is caused by. There is a need to ignore the rumors, theories, assumptions, and suspicions and get to some solid facts. Often a simple misunderstanding of some performance figure leads people to think there is a problem when there is no problem at all.

Note

There is a clear tendency to first blame the hardware.

A classic example is monitoring CPU I/O wait time and assuming high I/O wait time on an 8 way SMP is bad and indicates a disk problem. In fact it is just a quirk about the way I/O wait is reported before AIX 4.3.3. A system can report 80 percent I/O wait time was and easily provide sub seconds response times. So, the performance problem is actually a lack of understanding of the performance figures.

Another example is running tests on disk speed and comparing machines. One site determined that the disks behaved very differently on two identical RS/6000 machines; therefore, the slow machine must have an adapter or disk problem. It turned out to be the system administrator had been changing AIX parameters (`minpout`, `maxpout`) in an effort to increase a backup speed but had forgotten to put these back to the default values on one machine. So, the performance problem was a lack of careful administration and leaping to conclusions without checking the details.

Another site tried comparing an RS/6000 disk to a competitors machine. The test was simple and proved the RS/6000 was two thirds of the speed at reading and writing files. But, they did not understand the test. It turned out they were using the `cp` command, which meant the file was cached in memory after the first time and so was a pure write to the disk test. It was then found that the two machines were rated about the same speed, but the RS/6000 CPUs was an eight way and the other machine a three way. The test was only

using a single command, which used only one CPU; therefore, it was only using 1 eighth of the RS/6000's CPU power compared to one third of the other machine. In addition, the competitors' machine was writing to a fine striped file system (well known for good write performance), but on the RS/6000, it was a RAID 5 disk configuration. Writing to RAID 5 was a known performance bottleneck, with the former RAID adapters that did not provide the fast-write cache option. When all of these issues were fixed, the RS/600 turned out to be three times faster than the competition. So, the performance problem was to not understand the so called simple test and not being aware of the performance impact of SMP systems and various disk configurations.

Another example is the *it is slow* problem, but no ones knows:

- What is slow?
- How was slow measured?
- What is acceptable?
- Has it changed or was it always slow?
- What has been changed to make it slow?

The only performance fact available is *everybody thinks there is a problem*! The first task in this situation is trying to workout who started saying *it is slow* and why. Many performance problems result from too many managers dutifully escalating but are not based on no technical facts.

Fact or Fiction

Ignore the rumors - Just stick to the verifiable performance facts.

11.5.2 Gathering the information

Before changing anything on the system or database, it is worth checking a few basic parameters to make sure that they are either the default value, a sensible number, and that we know when, by whom, and why they have been changed.

11.5.2.1 Machine details

You should have an understanding of the machine including the following:

- CPU rating and number of CPUs
- Memory size and type
- Disk types and configuration
- Disk speeds in seek time and throughput

- Disk usage and logical volume
- Adapter types and ratings

11.5.2.2 Workload details

You should know the workload profile:

- The busy periods of the system so you can tune for these peaks in workload.
- The number of users logging on and actually being busy during the peak.
- The number of transactions per minute during the peaks.
- If there are online and batch peaks in the workload.

11.5.2.3 AIX virtual memory parameters

Check to see if the `vmtune` command is present on the system. See Appendix A, “AIX performance tools summary” on page 353 for more details. The `vmtune` command is part of the bos.adt.samples AIX fileset.

If is not available, then no one can have changed the `vmtune` parameters. If it is present, then use the command `vmtune` with no parameters to detail the current settings. Then, check that all the values are set to their default values. The default values for various levels of AIX and those that depend on the system size (for example memory) are documented in the AIX manuals.

If any parameter is not set to the default value, either:

1. Clearly document in the system log who, when, and why the parameter was set.
2. If no explanation is available, then you should seriously consider setting it back to the default value, as inappropriate values can have serious impact on AIX and database performance.

Inappropriate use of these parameters is the cause of many reported performance problems.

11.5.2.4 AIX system tunable parameters

Use the `lsattr -E -l sys0` command to detail the AIX tunable parameters.

Check to see that all the values are set to their default values. The default values for various levels of AIX and those that depend on the system size (for example, memory) are documented in the AIX manuals.

If any parameter is not set to the default value, either:

1. Clearly document in the system log who, when, and why the parameter was set.
2. If no explanation is available, then you should seriously consider setting it back to the default value, as inappropriate values can have serious impact on AIX and database performance.

Inappropriate use of these parameters is the cause of many reported performance problems. See Appendix A, “AIX performance tools summary” on page 353 for more details.

11.5.2.5 Network parameters

Use the `no -a` command to document the network parameters. See Appendix A, “AIX performance tools summary” on page 353 for more details.

11.5.2.6 Hardware configurations

Use the `lscfg` command to document the adapters. See Appendix A, “AIX performance tools summary” on page 353 for more details.

11.5.2.7 Document the RDBMS parameters

Use the DBA tools to output the database parameters currently in use. See Chapter 10, “Monitoring an RDBMS system for performance” on page 203 for more information.

11.5.3 Check for errors

The first thing is to see if the system is, in fact, trying to tell you that there is a problem. Many times the machines are in a computer room, and there are warning messages on the console screen that have gone unnoticed. In AIX, the second place to look is in the AIX system error log. To do this, use the commands: `errpt | pg`, and if there are recent errors, check the full details with `errpt -a | pg`. The error log is particularly likely to show up disk and network errors. If it is a network problem, then the network specialists or support group should be handed the problem. If it is a disk error, take actions immediately to rectify the problem. AIX reports temporary disk errors when a disk is about to fail but still works after the disk is stopped and started again by a *hardware reset*. This takes time and can produce a performance problem. If you are quick, the data can be moved to an alternative disk and, thus, a more serious problem can be avoided. Second, check the RDBMS error logs. On a DB2 UDB system, these are in the $DIAGPATH/db2diag.log file. On Oracle, these are in the $ORACLE_HOME/rdbms/logs directory.

11.5.4 Upgrade to the latest fix levels

Check if there are outstanding upgrades for:

- AIX - PTF
- Hardware firmware
- RDBMS fixes or PTF
- Application fixes or newer versions

Review what is available and decide when to schedule the upgrades.
Stay current: Latest release, or latest release -1 plus all fixes.

11.5.5 Investigating the system

This is an extension of the investigation order suggested in the redbook *Understanding IBM RS/6000 Performance and Sizing*, SG24-4810. It is different as this concentrates on an RDBMS system. The investigations are ordered into a sensible list of tasks.

First, check for errors and then go through CPU, memory, disks, and finally, the network. From experience, the fields memory and disks are more likely to be the bottleneck and the areas in which we have more choices in tuning. Run the following during a busy period and save the output:

- `vmstat`
- `iostat`
- Load perfpmr from the AIX media or the IBM ftp site (14.1, “Perfpmr - the performance data collection tool” on page 345)
- Run `_config.sh` from perfpmr
- `lsattr`
- `vmtune`
- If available, use other tools for performance monitoring, such as `nmon`.
- Use `lsvg`, `lspv`, and `lslv` to draw up a diagram of the disk configuration.
- Use the database tools to document the parameters and performance numbers.
- Use the applications to document the number of transactions.

11.5.6 Check and set top performance parameters

This is detailed in Chapter 12, “DB2 UDB tuning” on page 259, and Chapter 13, “Oracle tuning” on page 291 for the two databases.

11.6 Bottlenecks, utilization, and resources

The base-line is that we only have hardware. This means we have to make the best use of the CPU, memory, adapters, disks, and avoid network limits.

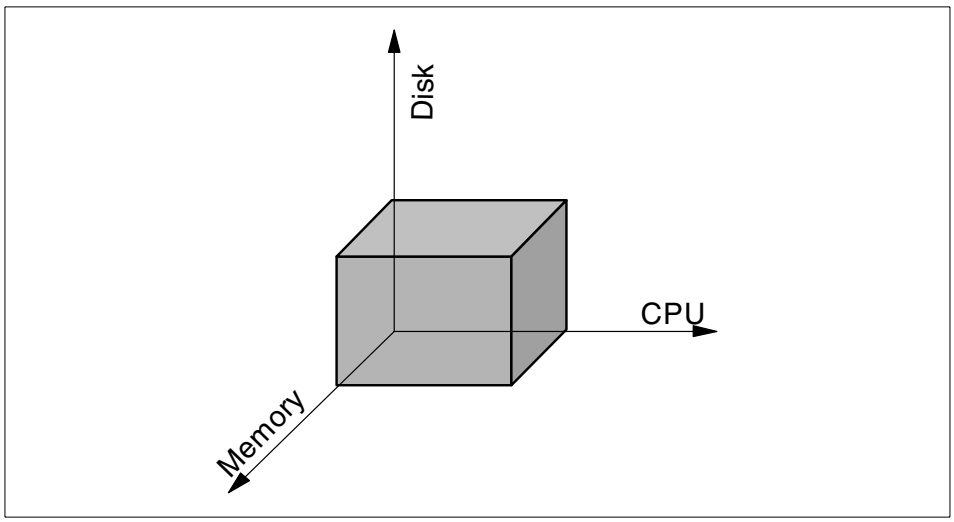


Figure 31. Hardware is: CPU, memory, and disks - Tuning means balancing

The CPU, memory, and disk of the system have to work together to achieve the maximum performance. In many of the things that can be tuned, we find that there is a trade off. For example, using more memory for disk block caching results in less disk I/O, and this means less CPU time for running device drivers. So, we are trading more memory used for less disk I/O. Figure 31 tries to how these things are all linked together. More memory can reduce disk I/O. More memory can mean more CPU due to longer in-memory searches. More disk I/O means more CPU to run the disk device drivers. All three dimensions are linked together.

The network is a further limiting factor, but we assume that the network is not the problem. This is a large subject area and is outside the scope of this redbook.

In a poorly balanced system, one of the components causes a bottleneck before the other components. Figure 32 shows the disks will hit a bottleneck before other components. If caching was increased, this would move the disk curve to the right and mean higher workloads are possible before the response time rises.

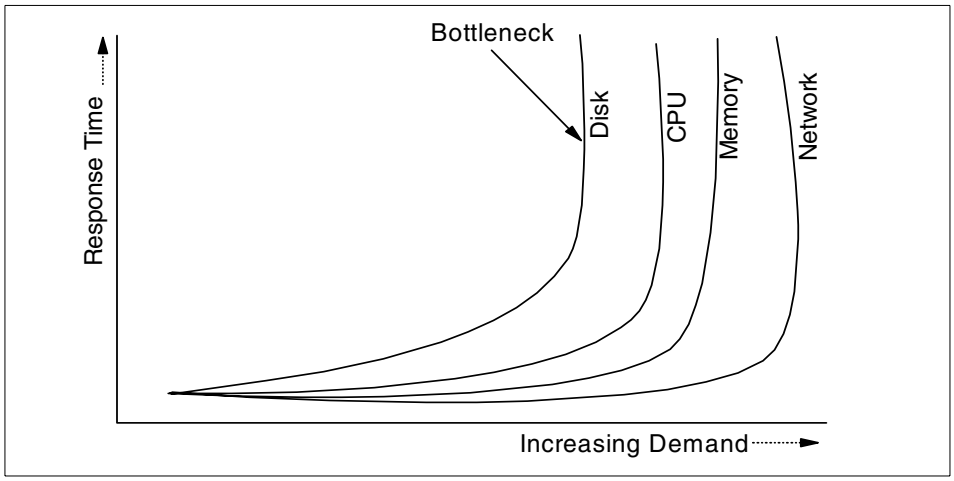


Figure 32. Poorly tuned means one bottleneck slows the system

In a well balanced and tuned system, we will find that no one component causes a bottleneck. For example, Figure 33 shows a well balanced system where no component is going to hold back the others.

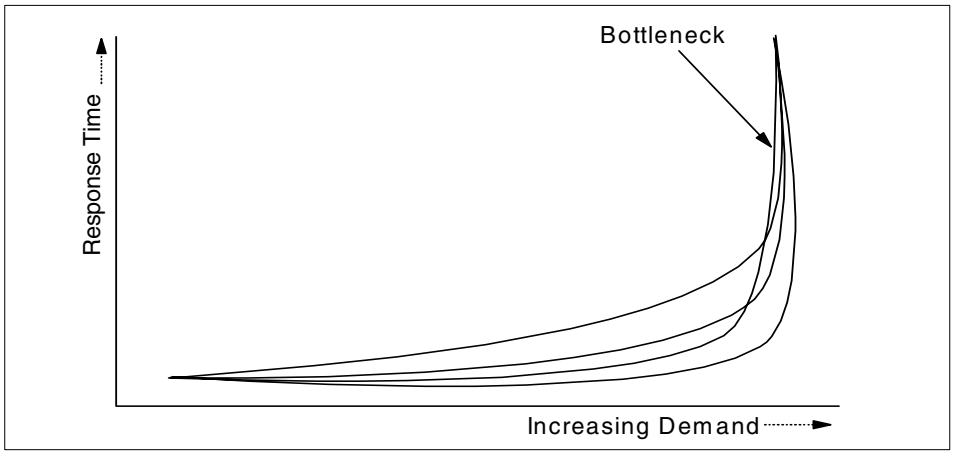


Figure 33. Well balanced systems postpone the bottleneck

11.6.1 Utilization goals

The first thing to investigate is the utilization level of each of these resources and compare these to the levels we would like to find. Each of these resources has an optimal level which, if exceeded, has a negative impact on

the overall system performance. These levels differ between workloads, and various people have different opinions on these levels too. Below is a table (Table 13) that you can use as a starting point.

Table 13. Bottleneck thresholds depend on the resource and workload

Resource	Measured by	OLTP	DSS	OLAP	Batch
CPU	Percent system + percent user time	70 percent	80 percent	70 percent	100 percent
System memory	See [1]	99 percent	99 percent	99 percent	99 percent
RDBMS memory	Cache and library hit ratio	99 percent	80 percent	99 percent	60 percent
Adapters	Percent busy and throughput	50 percent	50 percent	50 percent	50 percent [2]
Disks	Percent busy	40 percent	40 percent	40 percent	60 percent [2]
Network	Transfers and throughput	30 percent	40 percent	30 percent	60 percent [2]

Notes:
[1] AIX makes use of all available memory once it is running for any length of time.
[2] Batch can stress the system higher than these levels, but checks need to be made in order to make sure that other workloads or users of these resources are not affected.

Generally speaking, batch and DSS workloads can use higher utilization levels because they are focused on throughput rather than response time.

11.6.2 Insufficient CPU and latent demand

On well performing machines with many users attached, the workload varies during the working day. Figure 34 shows the typical pattern where there is a dip in workload during the lunch time period, and the CPU is not 100 percent busy during the peaks during mid-morning and mid-afternoon.

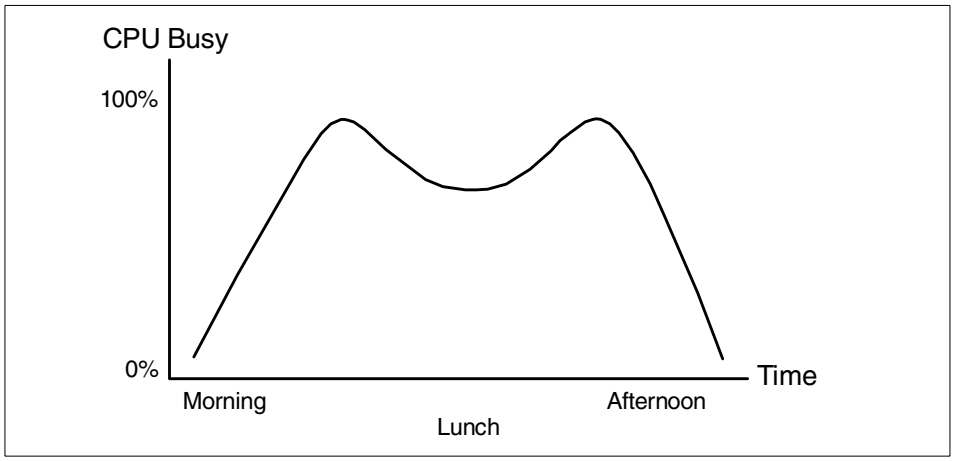


Figure 34. Demand over the working day

On an overworked system, the picture changes quite a lot because during the peaks, the CPU becomes 100 percent busy. It is nearly impossible to work out how much CPU power is required to stop the CPU from becoming the bottleneck. If the system is then tuned, the CPU may still be the bottleneck even though the tuning might improve the performance on the machine.

.

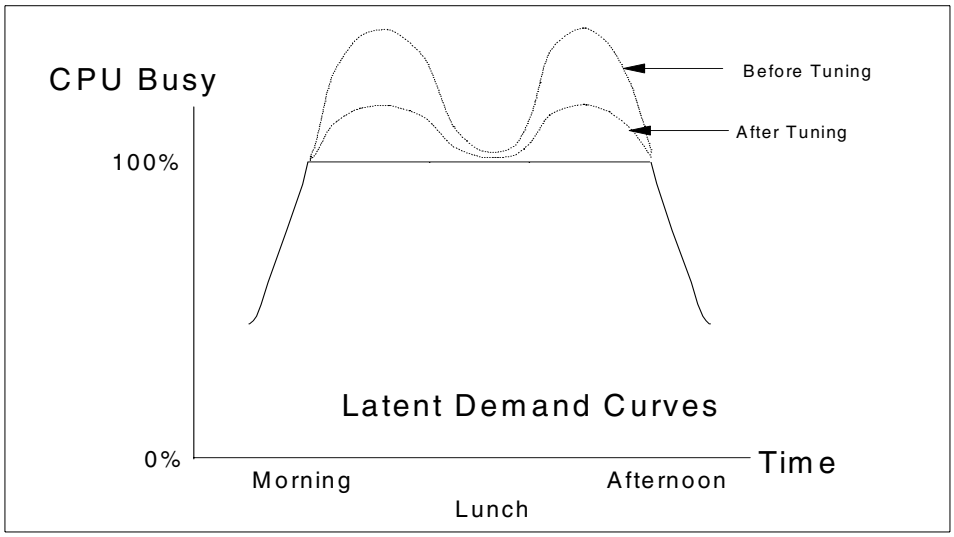


Figure 35. 100 percent busy machine means it is hard to determine if tuning helped or not

In these cases, the response times may improve even if the CPU is still overworked. If the CPU is still overworked after tuning, and an upgrade is recommended, it is still going to be very hard to estimate the CPU requirements of the upgraded machine because the height of the peaks cannot be determined.

11.6.3 Insufficient memory

When a machine does not have sufficient memory, the symptoms can be difficult to clearly detect. First, the machine might look like it has a disk or I/O throughput problem. This can be caused by simple UNIX paging and swapping activity. If the paging space is on a dedicated disk, this can be easily spotted. If the paging space is on other disks, such as the AIX, RDBMS code, or other working file disks, it can be difficult to determine that the high disk activity is due to paging. AIX commands, such as `vmstat`, can highlight paging activity. As the RDBMS buffer cache or pool takes up memory, one way is to reduce its size in order to free memory. This may stop paging but may mean the database cannot keep enough of the database in memory for high performance, and this results in the database performing a lot of extra I/O operations. This means paging I/O and database I/O have to be balanced, and a compromise has to be reached as shown in Figure 36.

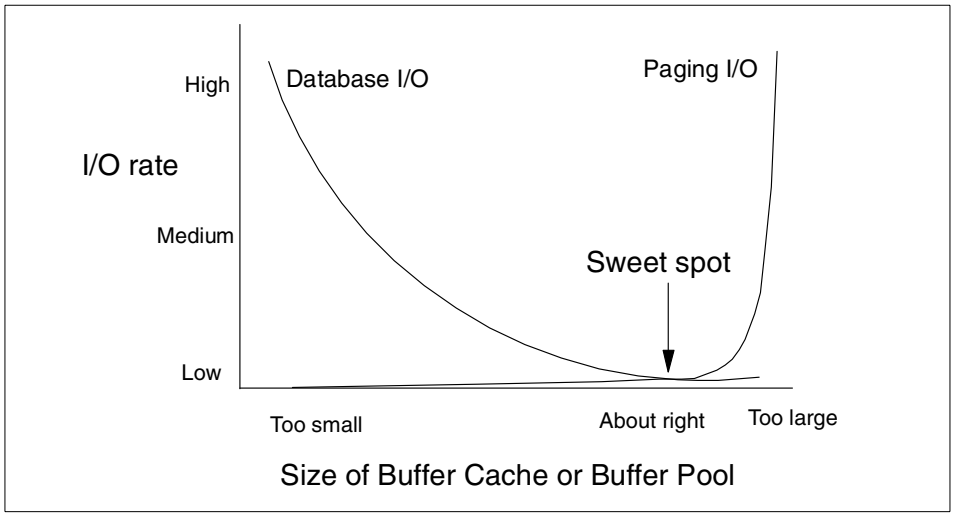


Figure 36. Balancing paging I/O against database I/O

Generally, most systems are tuned for near zero paging. On large systems, such as the RS/6000 H and S Series machines, sometimes paging cannot be avoided if a large number of user processes change their memory usage over

time. Given a well placed paging space, this does not present a problem and these machines are designed for efficient I/O subsystems in order to handle paging I/O.

To determine if the database buffer cache or buffer pool is of sufficient size, each database provides tools to provide details of the cache hit ratio. For OLTP systems, this is normally in the region of 95 percent to 99 percent. The other main usage of the memory by the RDBMS is for the shared_pool and log buffers.

11.6.4 Insufficient disk I/O

If the database does not have sufficient disk I/O capability, this is clearly seen by monitoring the disk performance statistics and CPU statistics. If the machine has high I/O wait CPU numbers, this *can* indicate I/O problems, but care has to be taken in trusting this number. First, I/O wait is assigned in a manner that is not clear to many people and has been changed in AIX 4.3.3 to make more sense. Please refer to Appendix A.22, “vmstat - Virtual Memory Management Statistics” on page 366 for more details.

If the disk statistics are investigated, then there are three areas to check:

- Disk I/O is distributed across the disk evenly. It is the job of both the system administrator and the database administrator to ensure the I/O is spread across many disks. If one disk is overworked, and the others under-used, then the whole system performance can suffer. The `iostat` command should be used to investigate this issue.

  Figure 37 shows a system with one disk overworked. This disk is probably slowing down the entire system.

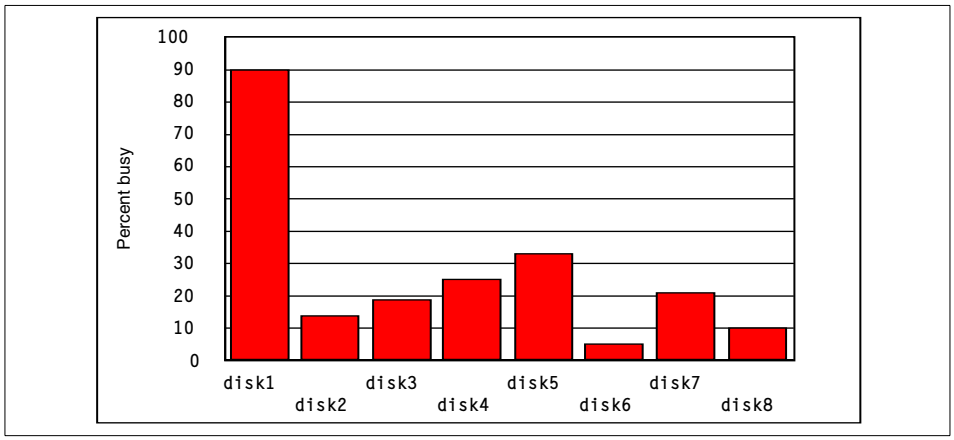


Figure 37. Unbalanced disk I/O - Disk 1 will cause a bottleneck

If data is moved from this disk and manually spread across other disks, the outcome should appear as that shown in Figure 38.

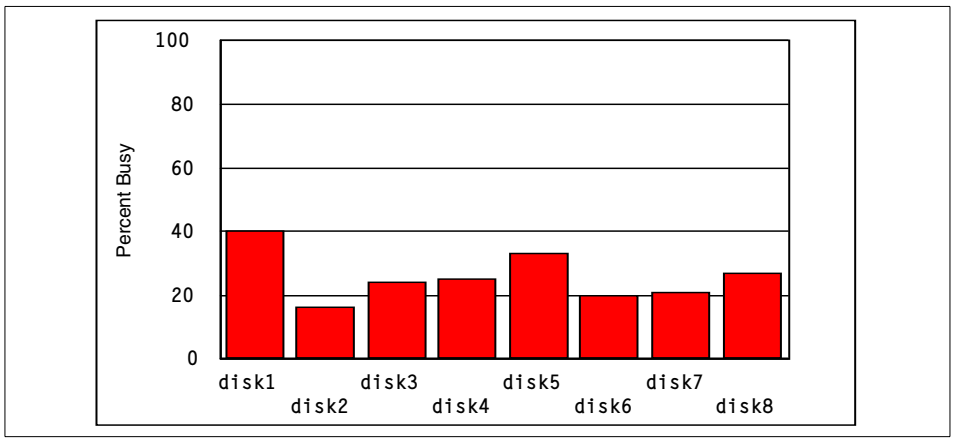


Figure 38. Balanced disk I/O - No bottlenecks

If, however, disk striping of some sort is used, the balancing of disk I/O across disks is performed by AIX or the disk subsystem (rather than by manually moving data). In this case, you will see very balanced I/O as shown in Figure 39.

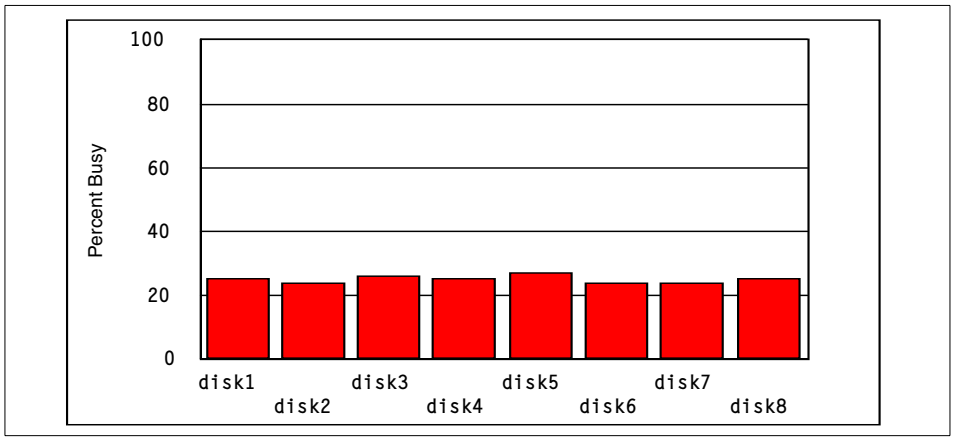


Figure 39. Disk I/O balanced by the system using data striping

- Disks are not overworked. For a disk to perform well and respond quickly to user demands, it is best to have the disk not running above 50 percent busy because it will mean that the queue for devices drivers can become large and, as a result, response times grow. For large query DSS and batch workload, the disks can be used above 50 percent busy. The `iostat` command should be used to investigate this issue.
- Disk channel is not a limiting factor. The disk channel is the PCI or MCA bus, SCSI, FCAL or SSA adapters, cables, and SSA loops. Any of these can become limiting factors especially when a lot of disks are attached on the same channel. It is very hard to track if the channel is overworked and to prove it is a limiting factor. The disks will appear to not be overworked, but the CPU appears to be in I/O wait state. The `nmon` tool gives the adapter busy and throughput statistics by adding up the statistics for all the disks attached to the adapter.

Note

The RDBMS logs (if placed on a dedicated disk or set of disks) are an exception to most of the rules above. Logs should not be spread across disks, are often overworked, and, if necessary, should have dedicated adapters and cables.

11.6.5 Insufficient network resources

The general rule of thumb is to monitor network throughput and do not allow it to be over 50 percent of the maximum throughput. Any collision detect

protocol network suffers with throughput problems above this level of network traffic.

11.6.6 Insufficient logical resource access

Within AIX and the RDBMS, access to logical resources is carefully controlled to make sure that data is not corrupted. This control is implemented as locks, latches, and semaphores. But whatever method is used, this restriction means that many operations cannot be performed in parallel, and for important resources, the tasks are serialized. This can have a large impact on performance.

Unfortunately, this is extremely hard to detect from observing CPU, memory, and disk activity. The system will seem to not be particularly busy, but the performance will be low. Only internal examination of AIX or the RDBMS will reveal the problem. AIX has trace facilities to allow this. DB2 UDB has the snapshot facility to allow this to be investigated. Oracle has numerous internal performance tables that can be used to monitor locks and latches. In all three cases, you may need assistance from technical support to determine the nature of the problem and provide a solution.

Fortunately, logical resource contention is one of the last places to investigate for possible performance problems.

11.7 What can we tune?

It is easy to think the only items we can change is a few disks and the database tuning parameters. But, there is quite a long list of things that can be changed. For example:

- A little purchasing power - Performance tuning should always highlight which component of the system should be considered for the next upgrade. If this is relatively inexpensive (similar to a few days performance tuning consultation), then it might be better to upgrade the machine immediately rather than continue tuning. Also, it allows planning and budgets to be prepared for longer term upgrades.
- Balancing the use of CPU, memory, and disk I/O - If one is overworked, you might be able to use the others to compensate.
- Balancing the use of memory between AIX, user processes, RDBMS processes, and the RDBMS shared memory.
- Balancing the various consumers of the RDBMS shared memory - For example: buffer, library, locks.

- Tuning disks by balancing speed over reliability with options, such as RAID 5, striping, and mirrors.
- Changing data placement on the disks via the AIX LVM options centre, middle, or edge.
- Removing hot spots by moving data between disks or hardware spreading of data via stripes or RAID 5.
- Dedicating disks to particular tasks for maximum response time and throughput. For example, the log disks.
- Ensuring equal use of disk and network I/O across adapters and that they are not approaching their theoretical or practical limits.
- Balancing disk I/O against memory. One of the many benefits of using memory is to reduce time-consuming disk I/O.
- Ensuring all CPUs of SMP machines are at work. Many performance problems are caused by a single batch process not making use of all the CPUs in the system.
- Maximizing backup rate to reduce the backup window or minimizing user interference with online backups. These considerations might effect the usage of disks and reserving disks for backup purposes.
- Balancing workloads. Many performance problems are simply poor management of workloads, in particular, batch operations or online requested reports. Sometimes users are willing to change their working habits if they realize they can improve performance (and their job) by making small changes to their work patterns.
- Identifying and fixing poor application modules and SQL statements.

  The database can help you work out the worst offenders. Many DBAs assume they cannot change the application code or SQL. This means they never investigate the SQL nor try to get it improved. Having identified the worst SQL example and the ones used repeatedly, these will yield the largest performance improvements:

  - If you know other sites that use the same application or SQL, then find out if they have the same list of issues.
  - Join the appropriate user groups.
  - Start providing feedback to the vendor or your own development team and start increasing the pressure for getting these problems fixed.
  - Although many developers resist making individual changes to application code, they do welcome real production feedback to improve their products performance in the longer term and for future releases.

- Starting to think in parallel on SMP machines. Unless you can make use of all the CPUs, you will be making use of a fraction of the available compute power.
- Starting to turn your attention toward logical resources, such as lock, spin counts, time-out, delay flags, and database latches, when the machine looks idle but responds badly.
- Finally, tuning the database via the parameters and options.

11.7.1 Tuning window

If this is a real production machine, there is often a problem to find:

1. The opportunity to change parameters or move disk space or data around
2. The means to run tests to check performance
3. A way to create a reproducible workload

Possible tuning windows are:

- At night, which makes tuning not very desirable to those doing the work.
- Just once per day when the database is restarted (perhaps after a backup), and you have to hope that you never make a mistake, therefore, making the system unusable in the morning.
- Sometimes a copy can be made (on similar or smaller size machine). If it has the same performance issues and architecture (for example, like an 8 way SMP), then this can be used to do a lot of tuning without effecting the production machine.
- Ideally in the pre-production phase, testing time is allocated to allow full scale, full speed testing, but time constraints and problems of user emulation can make this impossible.
- Extra machines can be rented, loaned, or a test run within IBM at a benchmark center, although this can be expensive in man-power terms.

11.8 Classic mistake list

These problems mostly arise due to people making simple statements but not covering the exceptions, that performance options do not change when hardware and software improves, and that their experience extends to all known cases.

Be very careful you do not encourage the folklore, misunderstandings, or over simplifications. The following lists the classic mistakes to avoid:

- Thinking file system based databases are as fast as raw devices.
- Thinking a RAID database is fast (it is inexpensive).
- Thinking RAID write performance will not be too bad or the database does not do many writes to disk.
- Thinking that using a new, just released, feature is going to work and fix all know performance issues first time with no testing.
- Misreading I/O wait CPU statistics and deciding a high value is a problem.
- Performing a seemingly simple test, not understanding the results, and assuming something is wrong with the machine.
- Assuming the database defaults will be fine (especially the Oracle init.ora parameters).
- Changing from a file system based database to raw devices and not adjusting the RDBMS buffer cache/pool to use more memory.
- Using the tool you know rather than the right tool.
- Changing AIX parameters to try and fix a problem, not knowing if they helped or not, and then forgetting to set them back to the safe default values.
- Running more than one database on a machine and expecting the machine to work out which is more important.
- Working out a transaction rate per second or minute, based on the number of transactions for a year in total, and assuming a completely flat and even work rate for five days a week and an even workload during the whole day. This is especially not true for e-business.
- Basing a transaction rate on marketing estimates.

Chapter 12. DB2 UDB tuning

For tuning a DB2 UDB database, there are some basic considerations to take, which include the following topics:

- Operational performance considerations
- Environmental considerations
- Application considerations
- System catalog statistics
- SQL compiler
- SQL explain facility
- Using the DB2 UDB governor
- Scaling the configuration

This chapter focuses on operational and environmental factors only because they can be changed during the life time of a database system and managed by the database administrator.

12.1 Performance improvement process

The following process is recommended to improve the performance of any system:

1. Establish performance indicators.
2. Define performance objectives.
3. Develop a performance monitoring plan.
4. Carry out the plan.
5. Analyze your measurements to determine whether you have met your objectives. If you have, consider reducing the number of measurements you make because performance monitoring itself uses system resources. Otherwise, continue with the next step.
6. Determine the major constraints in the system.
7. Decide where you can afford to make trade-offs and which resources can bear additional load. (Nearly all tuning involves trade-offs among system resources and the various elements of performance.)
8. Adjust the configuration of your system. If you think that it is feasible to change more than one tuning option, implement one at a time. If there are

no options left at any level, you have reached the limits of your resources and need to upgrade your hardware.

9. Return to Step 4 above and continue to monitor your system.

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

1. Return to Step 1 above.
2. Reexamine your objectives and indicators.
3. Refine your monitoring and tuning strategy.

Note

DB2 UDB provides many tools and commands that interface with the control files and catalog tables, thus, providing an easy, fast, and valuable monitoring information usually not well exploited on other RDBMSs.

12.2 General tuning elements

The following sections provide a short description of all tuning considerations of a DB2 UDB RDBMS.

12.2.1 Operational performance considerations

These considerations are about how to improve the operational performance by analyzing the performance indicators collected at run-time. The database administrator is responsible for maintaining a stable and well tuned RDBMS in order to achieve an excellent operational performance.

12.2.2 Environmental considerations

These considerations apply to database manager configuration parameters, database configuration parameters, such as buffpage or sortheap, and DB2 registry variables of DB2 UDB, and may be tuned by the database or system administrator.

12.2.3 Application considerations

A number of factors exist that can impact the runtime performance of an application, such as concurrency, locking, or stored procedures. Application developers have to pay attention to all these factors during application design and development because they can not be changed without recompiling and binding the program. However, if only the isolation level is changed, no recompilation is needed. In this case, only a bind with the new isolation level is required.

12.2.4 System catalog statistics

The database manager creates and maintains two sets of system catalog views: SYSCAT and SYSSTAT. These system catalog views are created when a database is created. The SYSCAT view is updated during normal operation in response to SQL data definition statements, environment routines, and certain utilities, such as RUNSTATS. Data in the system catalog views is available through normal SQL query facilities. The SYSSTAT view contains statistical information used by the optimizer. The optimizer uses these statistics to estimate the costs of alternative access paths that could be used to resolve a particular query. Some columns in the SYSSTAT views may be changed to investigate the performance of hypothetical databases. After a certain time or after a large number of rows were inserted or updated into a database, it is recommended to issue the `RUNSTATS` command so that all statistics will reflect the current, new state. The `RUNSTATS` command is a DB2 UDB command that updates statistics about the physical characteristics of a table and the associated indexes. These characteristics include the number of records, number of pages, and average record length. The optimizer uses these statistics when determining access paths to the data. It is recommended to call this command when a table has had many updates or after reorganizing a table. See the *IBM DB2 UDB Command Reference*, SC09-2844, for more details.

12.2.5 SQL compiler

The SQL compiler generates, during runtime of a *dynamic SQL* statement, the access plan for this particular query. It performs several steps before producing an access plan that can be executed. Information about access plans for *static SQL* is stored in the system catalog tables. When the package for a certain *static SQL* statement is executed, the database manager will use the information stored in the system catalog tables to determine how to access the data and provide results for the query. For more information on how the SQL compiler generates the access plan see, Chapter 20 of the *DB2 UDB Administration Guide.*

12.2.6 SQL Explain facility

The SQL Explain facility is part of the SQL Compiler that can be used to capture information about the environment in which the static or dynamic SQL statement is compiled. The information captured allows the database administrator to understand the structure and potential execution performance of SQL statements. The Explain facility assists in designing application programs, determines when an application should be rebound, and assists in database design (see 10.1.1.5, “Explain” on page 208).

12.2.7 Using the DB2 UDB governor

The governor monitors and changes the behavior of applications that run against a database. The governor is a DB2 UDB process (daemon) that collects statistics about the applications running against a database. It then checks these statistics against the rules that the database administrator specified in a governor configuration file that applies to that specific database. The governor then acts according to these rules. It can change the priority of a DB2 UDB process, or it can force an application that uses more resources as defined in the governor configuration file. The database administrator uses a front end utility to configure the governor. For more information, refer to Chapter 8 of the *IBM DB2 UDB Administration Guide: Performance*, SC09-2840.

12.2.8 Scaling the configuration

If a database system grows and reaches a configuration that does not satisfy the business need anymore, further system resources may be added. Please refer to 7.5, “System resource utilization” on page 131 for more information about the system resource demands. Some facts must be taken into consideration depending on the current database system. For example, the current database system is a single-partition configuration with a single processor (UP system). An exchange of the UP CPU planar through a SMP CPU planar allows the database manager to take advantage of the new processors by using the parallel implementations of DB2 UDB. However, to allow the new functionality, some configuration parameters should be reviewed and perhaps updated. These are:

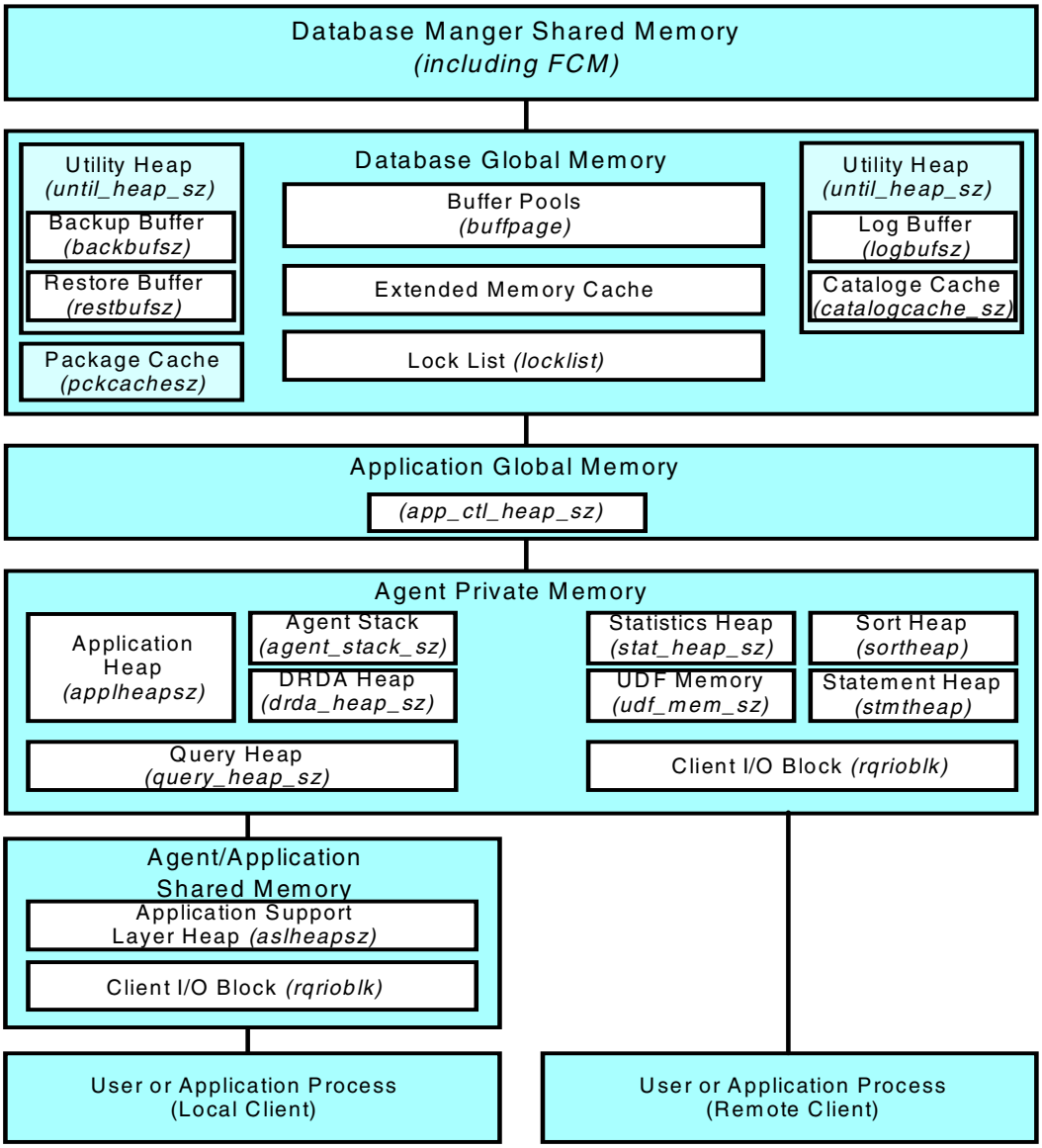
- Enable intra-partition parallelism (intra_parallel)
- Default degree (dft_degree)
- Maximum query degree of parallelism (max_querydegree)

Many more options exist for upgrading an SMP system with a single database partition to an SMP system with more CPUs and more database partitions. For more information about scaling a database system, refer to Chapter 9 of the *IBM DB2 UDB Administration Guide: Performance*, SC09-2840.

12.2.9 Memory usage by DB2 UDB

Many of the configuration parameters available in DB2 UDB affect memory usage on the system. To better understand how DB2 UDB manages the systems memory, the following figure shows how DB2 UDB uses memory for the database manager shared memory, for database global memory, for application global memory, for application private memory, and for

agent/application shared memory (see Chapter 23 of the *DB2 UDB Administration Guide).*



Note: Box size does not indicate relative size of memory.

Figure 40. Memory usage by DB2 UDB Database Manager

A database administrator should seek for balancing the overall memory usage on the system. Different applications that run on the operating system

may use memory in different ways. For example, some applications may use the file system cache, while the Database Manager uses its own buffer pool for data caching instead of the operating system facility.

12.3 What can you change to make a difference?

After creating an instance and a database, all configuration parameters come up with default values. Default values are implemented for systems with small databases and a relatively small amount of memory. These default values are not always sufficient for all installations. Different types of applications and users have different response time requirements. An OLTP system needs other tuning procedures than a DSS system. In an RDBMS environment, there are many factors affecting the performance of the entire system. Most of these parameters can be changed (configurable parameters), and some cannot be changed (informational parameters). DB2 UDB has been designed with a wide array of tuning and configuration parameters. These parameters can be subdivided in two general categories:

- Database Manager parameters
- Database parameters

Each parameter has a different scope. To understand the scope of a parameter allows the database administrator to estimate the potential impact of changing the parameter. The different scopes are:

- Database instance (for example, dir_cache)
- Database (for example, util_heap)
- Application/Agent (for example, sortheap)

The impact on performance of these parameters are different. They are categorized into four levels:

- High: This parameter can have a significant performance impact and should be selected and changed very carefully.
- Medium: This parameter indicates that it has some impact on performance.
- Low: This parameter has only a low impact.
- None: This parameter does not have to be observed because it does not directly impact performance.

All configurable parameters can be viewed, changed, or reset by any of the following methods:

- Using the DB2 UDB Control Center. This utility provides a comfortable and easy-to-use interface for database administration. It is available on the server itself but also on any supported client.
- Using the Command Line Processor (CLP). This is a non-GUI tool that provides the capability of issuing all available DB2 UDB commands from the AIX command line.
- Using the application programming interface (API) delivered with DB2 UDB.

12.4 What are the options?

This section shows the database manager, database, and environment configuration parameters that have an important impact on database performance.

12.4.1 Database manager configuration parameters

Database manager configuration parameters are related to the entire instance and reside on both database servers and clients. However, only few parameters are available on a client. They are subsets of database manager configuration on the server. Most of them either affect the amount of system resources that will be allocated, or they configure the setup of the database manager and the different communication subsystems.

The database manager configuration parameters are stored in a file named *db2systm*. It is created at the time of DB2 UDB instance creation and resides in the $HOME/sqllib directory. This file cannot be changed directly. After changing any of the database manager configuration parameters, the database manager must be stopped and restarted to make the changes available.

The following table shows the most important configurable database manager parameters with high and medium impact of performance. For a complete list of dbm parameters see Table 54 in Chapter 28, "Configuring DB2 UDB" of the *DB2 UDB Administration Guide.*

Table 14. Database manager configuration parameters at a glance

Database Manager parameter	Performance Impact	Description
agentpri	High	Priority of agents given to all database processes
aslheapsz	High	Applications support layer heap size

Database Manager parameter	Performance Impact	Description
audit_buf_sz	High	Audit buffer size
dos_rqrioblk	High	DOS requester I/O block size for DOS and Win 3.1 clients
fcm_num_anchors	High	Number of FCM message anchors
fcm_num_buffers	High	Number of FCM buffers
fcm_num_connect	High	Number of FCM connection entries
fcm_num_rqb	High	Number of FCM request blocks
intra_parallel	High	Enable intra-partition parallelism
java_heap_sz	High	Maximum Java interpreter heap size
max_querydegree	High	Maximum query degree of parallelism
num_poolagents	High	Agent pool size
rqrioblk	High	Client I/O block size
sheapthres	High	Sort heap threshold
backbufsz	Medium	Default backup buffer size
comm_bandwidth	Medium	Communications bandwidth
conn_elapse	Medium	Connection elapse time
dft_monswitches	Medium	Default database system monitor switches
dir_cache	Medium	Directory cache support
discover	Medium	Discovery mode
federated	Medium	Federated database system support
indexrec	Medium	Index re-creation time
initdari_jvm	Medium	Initialize DARI process with JVM
keepdari	Medium	Keep DARI process indicator
maxagents	Medium	Maximum number of agents
maxcagents	Medium	Maximum number of concurrent agents
max_connretries	Medium	Node connection retries
max_coordagents	Medium	Maximum number of coordinating agents

Database Manager parameter	Performance Impact	Description
maxdari	Medium	Maximum number of DARI processes
max_time_diff	Medium	Maximum time difference among nodes
maxtotfilop	Medium	Maximum total files open per application
min_priv_mem	Medium	Minimum committed private memory
num_initagents	Medium	num_initagents
num_initdaris	Medium	Initial number of fenced DARI processes in pool
priv_mem_thresh	Medium	Private memory threshold
query_heap_sz	Medium	Query heap size
restbufsz	Medium	Default restore buffer size
spm_log_path	Medium	Sync point manager log file path

12.4.2 Database parameters

These parameters reside only on a database server and are individually assigned to each database. The information is stored in the file SQLDBCON located under the directory SQLnnnnn (nnnnn is a number assigned when the database is created). It cannot be edited directly.

Updates to database configuration parameters do not take effect while applications or users are connected to the database. After termination of all applications and reconnection, the changes are available. If a database is in *active* state, it must be deactivated, which can be achieved by executing the `DEACTIVATE DATABASE` command or disconnecting all applications from the database.

Table 15 on page 267 lists database configuration parameters with high and medium performance impact. For the complete list of database parameters, see Table 56 in Chapter 12, "Configuring DB2 UDB" of the *IBM DB2 UDB Administration Guide: Performance*, SC09-2840.

Table 15. Database configuration parameters at a glance

Database parameter	Performance impact	Description
buffpage	High	Buffer pool size
avg_appls	High	Average number of active applications

Database parameter	Performance impact	Description
chngpgs_thresh	High	Changed pages threshold for asynchronous page cleaners
dft_degree	High	Default degree for intra-partition parallelism
logbufsz	High	Log buffer size
mincommit	High	Number of commits to group
num_iocleaners	High	Number of asynchronous page cleaners
num_ioservers	High	Number of I/O servers
pckcachesz	High	Package cache size
seqdetect	High	Sequential detection flag
sortheap	High	Sort heap size
locklist	High 1)	Maximum storage for lock list
maxlocks	High 1)	Maximum percent of lock list before escalation
app_ctl_heap_sz	Medium	Application control heap size
applheapsz	Medium	Application heap size
audit_buf_sz	Medium	Audit buffer size
catalogcache_sz	Medium	Catalog cache size
dbheap	Medium	Database heap
dft_extent_sz	Medium	Default extent size of table spaces
dft_loadrec_ses	Medium	Default number of load recovery sessions
dft_prefetch_sz	Medium	Default prefetch size
dft_queryopt	Medium	Default query optimization class
discover_db	Medium	Discover database
dlchktime	Medium	Time interval for checking deadlock
estore_seg_sz	Medium	Extended storage memory segment size
indexrec	Medium	Index re-creation time
locktimeout	Medium	Lock timeout
logfilsiz	Medium	Size of log files

Database parameter	Performance impact	Description
logprimary	Medium	Number of primary log files
logsecond	Medium	Number of secondary log files
maxappls	Medium	Maximum number of active applications
maxfilop	Medium	Maximum database files open per application
num_estore_segs	Medium	Number of extended storage memory segments
softmax	Medium	Recovery range and soft checkpoint interval
stmtheap	Medium	Statement heap size

12.4.3 DB2 UDB registry variables

Apart from the database manager and database configuration parameters, DB2 UDB provides a set of environment variables that are stored in the DB2 UDB profile registry. The `db2set` command allows the database administrator to display, set, or remove these profile variables. Some of these variables affect the performance of a database system.

Table 16. DB2 UDB environment variables affecting performance

Parameter	Description
DB2_AVOID_PREFETCH	Specifies whether or not prefetch should be used during crash recovery
DB2_BINSORT	Enables a new sort algorithm that reduces the CPU time and elapsed time of sorts.
DB2CHKPTR	Specifies whether or not pointer checking for input is required.
DB2_DARI_LOOKUP_ALL	Specifies if DB2 UDB server will perform a catalog lookup for ALL DARIs and stored procedures
DB2MEMDISCLAIM	Changes the behavior of DB2 UDB how to disclaim some or all memory
DB2MEMMAXFREE	Specifies the amount of free memory that is retained by each DB2 UDB agent
DB2_MMAP_READ	Allows DB2 UDB to use mmap as an alternate method of I/O
DB2_MMAP_WRITE	Used in conjunction with db2_mmap_read to allow DB2 UDB to use mmap as an alternate method of I/O
DB2_OVERRIDE_BPF	Specifies the size of the buffer pool, in pages, to be created at database activation, or first connection, time

Parameter	Description
DB2PRIORITIES	Controls the priorities of DB2 UDB processes and threads
DB2_RR_TO_RS	Set the Repeatable Read isolation level to Read Stability
DB2_SORT_AFTER_TQ	Specifies how the optimizer works with directed table queues in a partitioned database
DB2_NO_PKG_LOCK	Allows the Global SQL Cache to operate without the use of package locks to protect cached package entries.

12.5 Which options will make a large difference?

The previous tables show that many configurable database manager parameters and database parameters exist, which effect the performance of a database manager and the databases that run on it. The following topics show the most important database manager configuration (dbm) and database configuration (db) parameters. These are parameters that have a large impact on performance and should, therefore, be tuned first.

12.5.1 Buffer pool size (buffpage)

The buffer pool is the area of memory where database pages (table rows or indexes) are temporarily read and manipulated. All buffer pools reside in global memory, which is available to all applications using the database. The purpose of the buffer pool is to improve database performance. Data can be accessed much faster from memory than from disk. Therefore, the more data (rows and indexes) the database manager is able to read from or write to memory, the better the database performance. One component of the database manager is the *Bufferpool Services (BPS)*. The BPS initializes the segments for buffer pool and extended storage within the memory when the first connection is made to a database or at the time the database is activated through the `db2 activate database` *`database_name`* command. The BPS is responsible for reading data and index pages from disk into memory and to write pages from memory to disk. The BPS will use the File Storage Manager or the Raw Storage Manager to get the pages depending on whether an SMS tablespace or DMS tablespace is used. When an application accesses a row of a table for the first time, the BPS places the page containing that row from disk into the buffer pool. The next time an application requests data, the buffer pool is checked first if the data is in this memory area. If the requested data is found in the buffer pool, the BPS does not need to read the data from disk. If the buffer pools are not large enough to keep the required data in memory, the BPS has to read new data from disk. Avoiding data retrieval from disk storage results in faster performance.

There are two ways to place pages into the buffer pool and to write it back to disk:

- Read/write operations done by a db2agent resulting in synchronous I/O operations.
- Read operations performed by the I/O servers (prefetchers) and write operations done by the page cleaners using asynchronous I/O.

If a db2agent needs data requested by a query, and it cannot find this data in the buffer pool, it reads the pages from disk. This synchronous I/O consumes time. To avoid this response time, DB2 UDB uses asynchronous I/O servers to read data ahead into the buffer pool. This is done by the prefetcher. It is recommended to configure at least one I/O server for each physical disk (see also 12.5.3, “Number of asynchronous page cleaners (num_iocleaners)” on page 274). For performance reasons, it is desirable to hit as many pages as possible in the buffer pool. The *buffer pool hit ratio* indicates the percentage of time that the database manager did not need to load a page from disk into memory. The greater the buffer pool hit ratio, the lower the frequency of disk I/O.

Each database has at least one buffer pool, IBMDEFAULTBP, which is created when the database is created. You can create more than one buffer pool and can assign each buffer pool to a certain tablespace. For example, it is possible to assign a buffer pool to a tablespace that contains data of a large table or to create a buffer pool for an index tablespace to hold all indexes in memory.

Note

The configuration of one or more buffer pools is the single most important tuning area since it is here that most of the data manipulations take place for applications connected to the database. This is valid for regular data only (except large objects and long field data).

Never leave this parameter on its default value.

Buffer pool performance should be tracked permanently.

Description:

The buffer pool size parameter for DB2 UDB on AIX has a default value of 1000 pages. The range of the *buffpage* parameter is from (2*maxappls) 524 to 288 pages. The *maxappls* parameter defines the maximum number of active applications (see Table 15 on page 267). Related parameters are:

- Database heap (dbheap)
- Number of asynchronous page cleaners (num_iocleaners)
- Changed pages threshold (chngpgs_thresh)

Please refer to step number 9 on page 273 for information on buffer pool page sizes.

Recommendations:

1. The buffpage parameter controls the size of a buffer pool when the `CREATE BUFFERPOOL` and `ALTER BUFFERPOOL` SQL statements were run with NPAGES -1; otherwise, the buffpage parameter is ignored, and the buffer pool will be created with the number of pages specified by the NPAGES parameter. To determine whether the buffpage parameter is active, issue:
   `SELECT BPNAME, NPAGES from SYSCAT.BUFFERPOOLS`
   Each buffer pool that has an NPAGES value of -1 uses the buffpage parameter.

   Important

   Instead of using the buffpage parameter, it is recommended to use the `CREATE BUFFERPOOL` and `ALTER BUFFERPOOL` SQL statements to create and change buffer pools and their sizes. This is helpful if more than one buffer pool is installed on the system.

2. Because the size of the buffer pool has a major impact on performance, consider the following factors to avoid excessive paging:
   - The amount of physical memory installed on the database server.
   - The size of memory used by other applications running concurrently with the database manager on the same machine.
3. In an OLTP environment, it is recommended to allocate as much as 75 percent of the system's memory that is left after taking out the memory required by the Operating system, applications running on the system, and memory for communication to the buffer pools under the following conditions:
   - There are multiple users connected to the database.
   - This system is used as a database server only.
   - The applications access the same data and index pages repeatedly.
   - There is only one database installed.

   Other key OLTP parameters are mincommit, num_iocleaners, pckcacgesz, and agentpri.

4. Particularly in OLTP environments, where there is typically a repetitive access to indexes, it is recommended to strive to have a buffer pool large enough to keep all indexes in memory.
5. In a DSS environment, it is recommended to allocate up to 50 percent of the left over memory to the buffer pool. More memory is required to the database sort parameters for large queries.
6. The buffer pool hit ratio should reach 100 percent. This can be monitored by DB2 UDB monitor utility.
7. For every buffer pool page allocated, some space is used in the database heap for internal control structures. This means, if you increase the buffpage size parameter, increase the dbheap parameter also. Each page in the buffer pool has a descriptor. This descriptor is an internal structure of about 140 bytes for each page in the buffer pool. For every 30 buffer pool pages, there is an additional one page overhead in the dbheap size.
8. The size of buffer pool is used by the optimizer in order to determine access plans. Therefore, after changing the value of this parameter, you have to rebind your applications. When selecting the access plan, the optimizer considers the I/O cost of fetching pages from disk to the buffer pool. In this calculation, the optimizer will estimate the number of I/Os required to satisfy a query. This estimate includes a prediction of buffer pool usage since additional physical I/Os are not required to read rows in a page that are already in the buffer pool. The I/O cost of reading the tables can have an impact on:
   - How two tables are joined.
   - Whether an unclustered index will be used to read the data.
9. The page size of a bufferpool can be 4 KB, 8 KB, 16 KB, or 32 KB. The page size of the bufferpool must match the page size of the tablespaces that are associated with it.

12.5.2 Number of I/O servers (num_ioservers)

DB2 UDB can activate prefetchers that read data and index pages into the buffer pool anticipating their need by an application (asynchronously). Prefetchers are also used by utilities, such as backup, restore, and load for asynchronous I/O. In most situations, these pages are read just before they are needed. To enable prefetching, the database manager starts separate threads of control, known as I/O servers, to perform page reading. As a result, the query processing is divided into two parallel activities: Data processing (CPU) and data page I/O. The I/O servers wait for prefetch requests from the CPU processing activity.

The database parameter num_ioservers specifies the number of I/O servers for a database. No more than this number of I/Os for prefetching and utilities can be in progress at any time.

Description:

The default value of num_ioservres is 3, and the range is from 1 to 255. Related parameters are:

- Default prefetch size (dft_prefetch_sz)
- Sequential detection flag (seqdetect)

Recommendations:

Because one I/O server can serve only one I/O device (disk), it is recommended to configure one or two more num_ioservers than the number of physical devices on which the tablespace containers reside. It is better to use additional I/O servers since there is a minimal overhead associated with each.

12.5.3 Number of asynchronous page cleaners (num_iocleaners)

Page cleaners are DB2 UDB processes that monitor the buffer pool and asynchronously write pages to disk before the space in the buffer pool is required by another database agent. This means that the agents will not wait for changed pages to be written out before being able to read a page. Pages that are changed by an UPDATE statement must be written to disk to store them permanently. These pages are called *dirty pages*. After the dirty pages are written to disk, they are not removed from the buffer pool unless the space they occupy is needed for other pages. Page cleaners ensure that db2agents will always find free pages in the buffer pool. If an agent does not find free pages in the buffer pool, it must clean them itself, and the associated application will have a poorer performance. Another purpose of page cleaners is to speed the database recovery if a system crash occurs. The more pages that have been written to disk, the smaller the number of log file records that must be processed to recover the database.

Description:

The default value for num_iocleaners is 1; the range is from 0 to 255.

Related parameters are:

- Buffer pool size (buffpage)
- Changed page threshold (chngpgs_thresh)

Recommendations:

If this parameter is set to 0, no page cleaners are started, and, as a result, the database agents will perform all of the page writes from the buffer pool to disk. This parameter can have a significant performance impact on a database stored across many physical storage devices since, in this case, there is a greater chance that one of the devices will be idle. If no page cleaners are configured, your applications may encounter periodic log full conditions. If the applications for a database primarily consist of transactions that update data, an increase in the number of cleaners will speed up performance.

In an OLTP environment, where many transactions are run against the database, it is recommended to set the value of this parameter to between one and the number of physical storage devices used for the database. Environments with high update transaction rates may require more page cleaners to be configured. This is valid also for database systems with large buffer pools.

In a DSS environment, that will not have updates, it is usual to set this parameter to 0. The exception would be if the query workload results in many TEMP tables being created. This can be determined by using the Explain utility. In this case, it is recommended to set the number of I/O cleaners to the number of disks that assigned to the TEMP tablespace.

You can use the command `get snapshot for bufferpools on database_name` to monitor the write activity information from the bufferpools in order to determine if the number of page cleaners must be increased or decreased. You should reduce the number of page cleaners if both of the following conditions are true:

- The number of buffer pool data writes is approximately equal to the number of asynchronous pool data page writes.
- The number of buffer pool index writes is approximately equal to the number of asynchronous pool index page writes.

However, you should increase the number of num_iocleaners parameter if either of the following conditions are true:

- The number of buffer pool data writes is much greater than the number of asynchronous pool data page writes.
- The number of buffer pool index writes is much greater than the number of asynchronous pool index page writes.

12.5.4 Changed pages threshold (chngpgs_thresh)

This parameter can be used to specify the level (percentage) of changed pages at which the asynchronous page cleaners will be started if they are not currently active. When the page cleaners are started, they will build a list of the pages to write to disk. Once they have completed writing those pages to disk, they will become inactive again and wait for the next trigger to start. Therefore, this parameter is connected to the num_iocleaners parameter.

Description:

The default value is 60 percent, and the range is from five percent to 99 percent of dirty pages to be written. Related parameters are:

- Number of asynchronous page cleaners (num_iocleaners)

Recommendations:

In an OLTP environment, you should generally ensure that there are enough clean pages in the buffer pool by setting the chngpgs_thresh value to be equal to or less than the default value. A percentage larger than the default can help performance if the database has a small number of very large tables.

In an DSS environment, these page cleaners are not used.

12.5.5 Sort heap size (sortheap)

The sortheap is a database configuration parameter. It is the amount of private memory allocated to each process connected to a database at the time of the sort. The memory is allocated only at the time of sort and deallocated after sorting has been finished. It is possible for a single application to have concurrent sorts active. For example, in some cases, a SELECT statement with a subquery can cause concurrent sorts. The larger the table to be sorted, the higher the value should be for this parameter. If the value is too large, then the system can force to page if memory becomes overcommitted.

Description

The default value is 256 pages, and the range is 16 pages to 524 288 pages. A related parameter is:

- Sort heap threshold (sheapthres)

Recommendations:

If the memory for sorts is too small, the database manager will create temporary sort tables, possibly on disk. This reduces the performance. The use of sort heap can be minimized by the appropriate defined indexes. It is recommended to increase the size of sortheap when frequent large sorts are required. This can be monitored with the Explain utility.

The memory for sortheap is allocated from the same agent private memory heap as the application heap, statement heap, and statistic heap.

Because the optimizer uses this parameter when determining whether or not to pipe a sort, it is necessary to rebind the application when the value is changed.

This is one of the most important areas to be tuned since a sort operation done in real memory can significantly improve performance. It is recommended that the remaining real memory that is not allocated to the AIX, applications, and other DB2 UDB memory structures, is allocated to the sort operations.

If there are more data to be sorted than memory space, merge phases will be required in order to finish the sort operation. A possible way to avoid this is to increase the sortheap parameter.

The `get snapshot for database database_name` command will provide two indicators that can be used for tuning the sortheap parameter:

- total sort time
- total sorts

It is recommended that you keep on increasing the sortheap parameter as long as both of the following conditions are true:

- You have real memory available.
- The result value of the equation *total sort time/total sorts* is decreasing.

12.5.6 Sort heap threshold (sheapthres)

This is a database manager configuration parameter. DB2 UDB uses this parameter to control the sum of all sortheap allocations of all applications in the instance. Therefore, this parameter impacts the total amount of memory that can be allocated across the database manager instance for sortheap.

The sheapthres parameter is used differently for private and shared sorts.

- For private sorts, this parameter is an instance-wide soft limit on the total amount of memory that can be consumed by private sorts at any given time. When the total private-sort memory consumption for an instance

reaches this limit, the memory allocated for additional incoming private-sort requests will be considerably reduced.

- For shared sorts, this parameter is a database-wide hard limit on the total amount of memory consumed by shared sorts at any given time. When this limit is reached, no further shared-sort memory requests will be allowed (until the total shared-sort memory consumption falls below the limit specified by sheapthres).

Description

The default value is 20,000 pages, and the range is from 250 to 2,097,152 pages.

A related parameter is:

- Sort heap size (sortheap)

Recommendations:

Explicit definition of the threshold prevents the database manager from using excessive amounts of memory for large numbers of sorts. It is recommended to set this value to a reasonable multiple of the largest sortheap parameter defined in the database manager instance. This parameter should be at least two times of the largest sortheap value for any database within the instance.

It is important to be aware that, when using DB2 UDB V5.2, and when the database manager parameter intra_parallel is enabled, the sheapthres does not simply work as a limiting number anymore, but the amount of space defined for this parameter is automatically allocated from memory.

12.5.7 Statement heap size (stmtheap)

The statement heap size is a database configuration parameter that specifies the size of workspace used for the SQL compiler during the compilation of an SQL statement. For dynamic SQL statements, this memory area will be used during execution of the application. For static SQL statements, it is used during the bind process. The memory will be allocated and released for every SQL statement only.

Description

The default value is 2,048 pages, and the range is from 128 to 60,000 pages.

Recommendations:

For most cases, the default value can be used. If an application has very large SQL statements, and DB2 UDB reports an error when it attempts to compile a statement, then the value of this parameter has to be increased. The error messages issued are:

- SQL0101N The statement is too long
- SQL0437W Performance of this complex query may be sup-optimal. Reason code 1.

These messages are sent to the applications that run the queries and are also logged in the DB2 UDB error log file called *db2diag.log*.

12.5.8 Package cache size (pckcachesz)

This database configuration parameter is used to define the amount of memory for caching static and dynamic SQL statements. Caching packages allows the database manager to reduce its internal overhead by eliminating the need to access the system catalogs when reloading a package or, in the case of dynamic SQL, eliminating the need for compiling a query twice. For example, if two users run the same application with the same query, the access strategy for this query can be used by both users as long as the compilation environment for both users and the application is the same. The compilation environment includes isolation levels, query optimization level, blocking, and application code page.

The package cache is available until the application terminates, or the cache runs out of space.

Description:

The default value is -1, which means the value used to calculate the page allocation is eight times the value specified for the maxappls configuration parameter. The range is from 32 to 64,000 pages.

Recommendations:

The package cache is important in an OLTP environment where the same query is used multiple times by multiple users within an application. To tune this parameter, it is helpful to monitor the package cache hit ratio. This value shows if the package cache is used effectively. If the hit ratio is large (> 90 percent), the package cache is performing well.

The package cache hit ratio can be obtained by the following formula:

(1 - (package cache inserts / package cache lookups)) * 100 percent

These indicators can be retrieved by the `get snapshot for database on database_name` command.

12.5.9 Database heap size (dbheap)

Each database has one memory area called a database heap. It contains control block information for tables, indexes, table spaces, and buffer pools. It also contains space for the event monitor buffers, the log buffer (logbufsz), and the catalog cache (catalogcache_sz). The memory will be allocated when the first application connects to the database and keeps all control block information until all applications are disconnected.

Description

The database heap size configuration parameter has a default value of 1,200 pages, and the range is from 32 to 60,000 pages. Related parameters are:

- Catalog cache size (catalogcache_sz)
- Log buffer size (logbufsz)

Recommendations:

Each page in the buffer pool has a descriptor of about 140 bytes. For every 30 buffer pool pages, an additional page for overhead is needed in the database heap. For databases with a large amount of buffer pool, it is necessary to increase the database heap appropriately.

12.5.10 Catalog cache size (catalogcache_sz)

The catalog cache is part of the database heap. It is used to store table descriptor information that is used when a table, view, or alias is referenced during the compilation of an SQL statement. When a transaction references a table, it causes an insert of a table descriptor into this cache so that subsequent transactions referencing that same table can use that descriptor and avoiding reading from disk.

Running any DDL statements against a table will purge that table's entry in the catalog cache. Otherwise, a table entry is kept in the cache until space is needed for a different table, but it will not be removed from the cache until any units of work referencing that table have completed.

Description

The catalog cache size parameter indicates the amount of memory used for the catalog cache. The default value is 64, and the range is from 1 to the size of database heap. Related parameters are:

- Database heap size (dbheap)
- Log buffer size (logbufsz)

Recommendations:

The default value is appropriate for most database environments. More cache space is required if a unit of work contains several dynamic SQL statements or if a package is bound to the database that contains a lot of static SQL statements. For OLTP databases with a large amount of tables and views, it is necessary to improve this value by tuning it using small increments. If the catalog cache hit ratio observed by DB2 UDB monitor utility is less than 80 percent, it is recommended to increase the value also.

12.5.11 Log buffer size (logbufsz)

This database configuration parameter specifies the amount of the database heap to use as a buffer for log records before writing these records to disk. The log records are written to disk when one of the following occurs:

- A transaction commits, or a group of transactions commit, as defined by the mincommit configuration parameter.
- The log buffer is full.
- As a result of some other internal database manager event.

Buffering the log records will result in more efficient logging file I/O because the log records will be written to disk less frequently, and more log records will be written at each time.

Description

The default value for this parameter is eight pages, and the range is from four pages to 512 pages. Related parameters are:

- Catalog cache size (catalogcache_sz)
- Database heap (dbheap)
- Number of commits to group (mincommit)

Recommendations:

It is recommended to increase the value of log buffer size if there is a high disk utilization on the dedicated disks for log files.

12.5.11.1 Maximum number of active applications (maxappls)

This database parameter specifies the maximum number of concurrent applications that can be connected (both local and remote) to a database. Since each application that attaches to a database causes some private memory to be allocated, allowing a larger number of concurrent applications will potentially use more memory.

Description:

The default value for this parameter is 40 active applications, and the range is from 1 to 64,000 active applications. Related parameters are:

- Maximum number of agents (maxagents)
- Maximum number of coordinating agents (max_coordagents)
- Maximum percent of lock list before escalation (maxlocks)
- Maximum storage for lock list (locklist)
- Average number of active applications (avg_appls)

Recommendations:

Increasing the value of this parameter without decreasing the maxlocks parameter or increasing the locklist parameter can cause that the database's limit on locks to be reached more frequently, thus, resulting in many lock escalation problems.

12.5.12 Maximum number of agents (maxagents)

This parameter indicates the maximum number of database manager agents (db2agent) available at any given time to accept application requests.There are two types of connections possible that require the use of DB2 UDB agents. Local connected applications require db2agents within the database manager instance as well applications running on remote clients. The maxagents parameter value must be at least equal to the sum of both values. This parameter is useful in memory constrained environments to limit the total memory usage of the database manager because each additional agent requires additional memory.

Description

The default value for maxagents is 200 agents, and the range is one to 64,000 agents.

Related parameters are:

- Maximum number of active applications (maxappls)
- Maximum number of concurrent agents (maxcagents)
- Maximum number of coordinating agents (max_coordagents)
- Maximum number of DARI processes (maxdari)
- Minimum committed private memory (min_priv_mem)
- Agent pool size (num_poolagents)

Recommendations:

The value of maxagents should be at least the sum of the values for maxappls in each database allowed to be accessed concurrently.

12.5.13 Maximum storage for lock list (locklist)

This parameter indicates the amount of storage that is allocated to the lock list. There is one lock list per database, and it contains the locks held by all applications concurrently connected to the database. Locking is required to ensure data integrity; however, too much locking reduces concurrency. Both rows and tables can be locked. Each lock requires 32 or 64 bytes of the lock list depending on whether other locks are held on the object:

- 64 bytes are required to hold a lock on an object that has no other locks held on it.
- 32 bytes are required to record a lock on an object that has an existing lock held on it.

If the memory assigned for this parameter becomes full, the database manager performs lock escalation. Lock escalation is the process of replacing row locks with table locks, thus, reducing the number of locks in the list. DB2 UDB selects the transaction using the largest amount of the locklist and changes the record locks on the same table to a table lock. Therefore, one lock (table lock) replaces many locks (record locks) of a table. This reduces the concurrency and, therefore, the performance. For example, two applications, A and B, select and update rows on a certain table. The lock manager manages the data access on row level. If the database manager has to change the locks for application A from row locking to table locking, the other application, B has to wait until application A releases the lock on that entire table. This can lead to hang situations from a user's point of view.

Description

The default value for this parameter is 100 pages, and the range is from four pages to 60,000 pages. Related parameters are:

- Maximum percent of lock list before escalation (maxlocks)
- Maximum number of active applications (maxappls)

Recommendations:

If lock escalations are causing performance or hang problems, it is recommended to increase the value of this parameter. Additional considerations that affect application design are:

- It is helpful to perform frequent `COMMIT` statements to release locks.
- When an application performs many updates, it is recommended to lock the entire table (using the SQL `LOCK TABLE` statement) before updating. This will use only one lock, keeps other applications from interfering with the updates, but does reduce concurrency of the data.
- The use of *Cursor Stability* isolation level decreases the number of share locks held. If application integrity requirements are not compromised, it is recommended to use *Uncommitted Read* instead of Cursor Stability to further decrease the amount of locking.

If many lock escalations are performed by the database manager, deadlocks between applications can occur, which will result in transactions being rolled back. Please refer to Appendix B.1.16, “Display the number of deadlocks and lock escalations” on page 373 for more information about monitoring deadlocks and lock escalations.

12.5.14 Maximum percent of lock list before escalation (maxlocks)

This parameter defines a percentage of the lock list held by an application that must be filled before the database manager performs escalation. When the number of locks held by any one application reaches this percentage of the total lock list size, lock escalation will occur for the locks held by that application.The database manager determines which locks to escalate by looking through the lock list for the application and finding the table with the most row locks. If after replacing these with a single table lock, the maxlocks value is no longer exceeded, lock escalation will stop. If not, it will continue until the percentage of the lock list held is below the value of maxlocks. The maxlocks parameter multiplied by the maxappls parameter cannot be less than 100.

Description

The default value for this parameter is 10 percent, and the range is from one percent to 100 percent.

Related parameters are:

- Maximum storage for lock list (locklist)
- Maximum number of active applications (maxappls)

Recommendations:

When setting maxlocks, the following formula helps to calculate the size of the lock list:

maxlocks = 100 * (512 locks per application * 32 bytes per lock * 2) / (locklist * 4096 bytes)

This sample formula allows any application to hold twice the average number of locks. You can increase maxlocks if few applications run concurrently since there will not be a lot of contention for the lock list space in this situation.

12.5.15 Maximum query degree of parallelism (max_querydegree)

This database manager parameter specifies the maximum degree of intra-partition parallelism that is used for any SQL statement executing on this instance of the database manager. An SQL statement will not use more than this number of parallel operations within a partition when the statement is executed. The intra_parallel configuration parameter must be set to YES to enable the database partition to use intra-partition parallelism.

Description

The default value is -1, which means any value, and that the database system (optimizer) determines the value of this option. The range is from one to 32,767 as value for degree.

Related parameters are:

- Default degree (dft_degree)
- Enable intra-partition parallelism (intra_parallel)

Recommendations:

This parameter can be used to change the degree of parallelism for an SQL statement that was specified at statement compilation time using the `CURRENT DEGREE` special register or specified with the `DEGREE` bind option. It is useful on an SMP database system only and should not exceed the number of CPUs of this system. Please refer to the *DB2 UDB Administration Guide: Performance V6,* which contains detailed information on query parallelism.

12.5.16 DB2MEMDISCLAIM and DB2MEMMAXFREE

Depending on the workload being executed and the pool agents configuration, it is possible to run into a situation where the committed memory for each DB2 UDB agent will stay above 32 MB even when the agent is idle. This behavior is expected and usually results in good performance, as the memory is available for fast reuse. However, on a memory constrained system, this may not be a desirable side effect. The db2set command

```
db2set DB2MEMDISCLAIM = yes
```

avoids this condition. This variable tells the AIX operating system to stop paging the area of memory so that it no longer occupies any real storage. This variable tells DB2 UDB to disclaim some or all memory once freed depending on DB2MEMMAXFREE. This ensures that the memory is made readily available for other processes as soon as it is freed.
DB2MEMMAXFREE specifies the amount of free memory that is retained by each DB2 UDB agent. It is recommended to set this value to 8 MB by using:

```
db2set DB2MEMMAXFREE = 8000000
```

12.5.17 DB2_PARALLEL_IO

This registry variable can be used to force parallel I/O for a tablespace that has a single container. When reading data from, or writing data to, tablespace containers, the database manager may use parallel I/O if the number of containers in the database is greater than 1. However, there are situations when it would be beneficial to have parallel I/O enabled for single container tablespaces. For example, if the container is created on a single RAID device that is composed of more than one physical disk, you may want to issue parallel read and write calls. The DB2_PARALLEL_IO variable can be set to one specific tablespace or to all tablespaces of that instance. For example, to enable parallel I/O for tablespace USERSPACE1 with tablespace ID 2, the command is:

```
db2set DB2_PARALLEL_IO= 2
```

12.5.18 DB2_STRIPED_CONTAINERS

When creating a DMS tablespace container (device or file), a one-page tag is stored at the beginning of the container. The remaining pages are available for data storage by DB2 UDB and are grouped into extent-sized blocks. When using RAID devices for tablespace containers, it is suggested that the tablespace is created with an extent size that is equal to, or a multiple of, the RAID stripe size. However, because of the one page container tag, the extents will not line up with the RAID stripes, and it may be necessary during an I/O request to access more physical disks than would be optimal. DMS table space containers can now be created in such a way that the tag exists in its own (full) extent. This avoids the problem described above, but it requires

an extra extent of overhead within the container. To create containers in this fashion, set the DB2 UDB registry variable DB2_STRIPED_CONTAINERS to ON by issuing:

```
db2set DB2_STRIPED_CONTAINERS=ON
```

Any DMS container that is created will have new containers with tags taking up a full extent. Existing containers will remain unchanged.

12.5.18.1 Tablespace page size

Since Version 5.2, DB2 UDB provides the possibility to expand the page size of tablespaces from default 4 KB to 8 KB. Version 6.1 supports 16 KB or 32 KB. This allows larger tables and larger row lengths. For example, a table created in a tablespace with a page size of 32 KB can reach a maximum size of 512 GB.

A bufferpool using the same page size must be assigned to each tablespace. For instance, if you have tablespaces with 4 KB, 8 KB, and 16 KB page sizes within a database, the database must have at least three bufferpools that also use a page size of 4 KB, 8 KB, and 16 KB.

A tablespace page size larger than 4 KB is advantageous for tables with large data rows. However, it is important to be aware that on a single data page there will never exist more than 255 rows of data, regardless of the page size.

12.5.19 Reorganizing tables

The performance of SQL statements that use indexes can be impaired after many updates, deletes, or inserts have been made. Newly inserted rows can often not be placed in a physical sequence that is the same as the logical sequence defined by the index (unless you use clustered indexes). This means that the Database Manager must perform additional read operations to access the data because logically sequential data may be on different physical data pages that are not sequential. The DB2 UDB `REORG` command performs a reorganization of a table by reconstructing the rows to eliminate fragmented data and by compacting information.

REORGCHK

Because the REORG utility needs a lot of time for reorganizing a table, it is useful to run the `REORGCHK` command before running the `REORG` command. The REORGCHK utility calculates statistics on the database to determine if tables need to be reorganized or not. Please refer to 7.7.1, “DB2 UDB reorganization method” on page 138 for more information about reorganizing the tables.

12.6 Simulating through SYSSTAT views

The DB2 UDB optimizer is responsible for estimating and generating an optimal access plan for any SQL query statement. To achieve an access plan it uses statistics of the database system. When optimizing SQL queries, the decisions made by the SQL compiler are heavily influenced by the optimizer's model of the database contents. This data model is used by the optimizer to estimate the costs of alternative access paths that could be used to resolve a particular query.

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

Examples of the statistics available that help define the data model to the optimizer include:

- The number of pages in a table and the number of pages that are not empty.
- The degree to which rows have been moved from their original page to other (overflow) pages.
- The number of rows in a table.
- The number of distinct values in a column.
- The degree of clustering of an index, that is, the extent to which the physical sequence of rows in a table follows an index.
- The number of index levels and the number of leaf pages in each index.
- The number of occurrences of frequently used column values.
- The distribution of column values across the range of values present in the column.
- Cost estimates for user-defined functions (UDFs).

Statistics for objects are updated in the system catalog tables only when explicitly requested. Some, or all, of the statistics may be updated by:

- Using the RUNSTATS utility (see 12.2.4, “System catalog statistics” on page 261)
- Using `LOAD` with statistics collection options specified

- Coding SQL UPDATE statements that operate against a set of predefined catalog views

With SYSSTAT, the database administrator is able to simulate a non-existent database or to clone an existent database into a test database. To do this, they can create a new database on a test system and then change the contents of the SYSSTAT tables with SQL UPDATE statements in order to achieve that the optimizer creates different access plans under different conditions. With the Explain utility, the database administrator can see which access plan the optimizer uses and which cost the optimizer estimates for a given SQL statement.

For example, to simulate the selection of rows from two large tables that do not really exist in this size, using the database described in Appendix D, the administrator first substitutes the entry for the CARD column in the SYSSTAT.TABLES with a large value.The DBA then changes the cardinality of table CUSTOMER from the current value to a fictitious value of 850,000 rows and for table ORDERS to a value of 5,000,000 rows with:

```
UPDATE SYSSTAT.TABLES SET CARD = 850000 WHERE TABNAME = 'CUSTOMER'

UPDATE SYSSTAT.TABLES SET CARD = 5000000 WHERE TABNAME = 'ORDERS'.
```

After that update, the administrator starts the SQL query and monitors its execution with the Explain tool. They can see how the optimizer changes the access plan and estimate the query cost. With this method, the administrator can estimate how an increasing amount of data affects the performance of that given query. The administrator can also clone their production database to a test database and can simulate different performance strategies. The DB2 UDB tool, db2look, is designed to capture all table DDLs and statistics of the production database to replicate it to the test system.

Note

The SYSSTAT views should only be updated manually for modeling a production environment on a test system or for *what-if* analysis. Statistics should not be updated on production systems.

Chapter 13. Oracle tuning

In this chapter, we cover the important items to make your Oracle database run well on the AIX platform. These tuning details are AIX specific, and we do not recommend using them on other platforms.

You can read this chapter in two ways:

- If you start from the top and read your way though, then you will follow the structured fine tuning approach discussed in 11.4, “Formal fine tuning method” on page 232.
- The alternative is jumping straight into the tuning hints and tips that start in 13.6, “Evaluate the top 10 Oracle parameters” on page 303 and then browse the details for particular issues you want to study further.

Tuning Oracle is a constant task. We have selected the top high-win performance issues and AIX specific issues to be included here. You will, no doubt, have your favorite hints, and your feedback would be welcome for the next revision of this redbook.

13.1 What can you change to make a difference?

There are several levels that need addressing in order to tune the Oracle RDBMS:

1. RS/6000 hardware design

   This is covered in Chapter 7, “Designing a system for an RDBMS” on page 121 and Chapter 8, “Designing a disk subsystem” on page 149.

2. AIX tuning parameters
3. Application design

   Not covered in this book (see comment below).

4. Use of SQL

   Not covered in this book (see comment below).

5. Disk configuration
6. Logical volume parameters
7. Oracle instance parameters in the init.ora file
8. Oracle dynamic tables

Note: Items 3 and 4 are not the prime focus of this book. This chapter will cover the other items and how to tune them for maximum Oracle performance with AIX.

13.2 Oracle tuning order

The Oracle Server Tuning manual recommends the following order for tuning:

1. Install and configuring Oracle
2. Application Design
3. Data Access Method (SQL and indexes)
4. Memory Allocation
5. Disk I/O
6. CPU usage
7. Resource Contention

After each item from number 3 and onwards, it is recommended that, if a change is made, the tuning starts again from the top. For example, if a change is made to the disk I/O, then the data access methods and memory allocation need to be re-checked. This is shown in a diagram form in Figure 41.

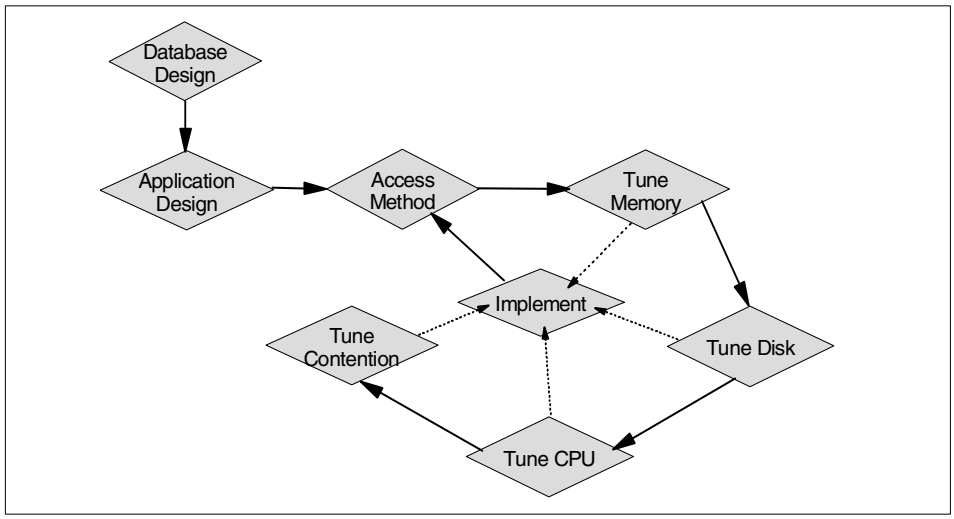


Figure 41. Oracle tuning sequence

Note: Application design and SQL tuning fall outside the scope of this book.

This is clearly a database-centric view of a system. We recommend the same tuning order, but highlight:

- That once the memory is adjusted, it will effect the disk I/O performance enormously.
- The disk I/O is the main area that has a lot of options and opportunities for tuning.
- The CPU usage can be measured, but there is not much that can be done to tune it apart from tuning other resources to avoid using the CPU.
- Tuning contention is really tuning Oracle internals and requires detailed understandings of the Oracle structure and algorithms.

If we remove the two design phases and add in the *change all at once* approach, which attempts to fix the major performance issues in one step, the tuning approach is that which is shown in Figure 42.

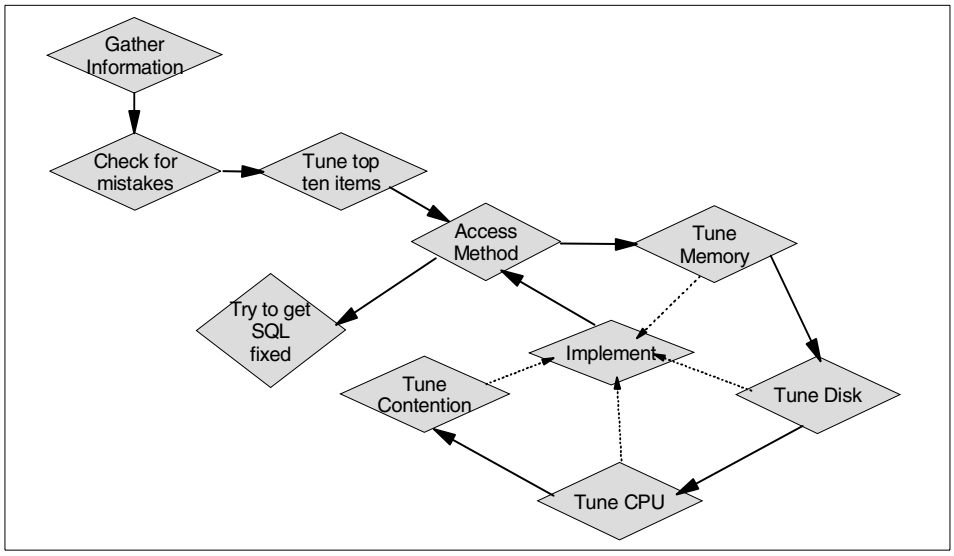


Figure 42. Change all at once plus practice tuning sequence

This practical sequence involves a number of stages:

- The first thing to do is gather all the information possible to build a picture of the machine and how it is behaving.
- Document the Oracle parameters.
  Use the DBA tools to output the Oracle parameters currently in use. This can be done in a number of ways:

- Using `svrmgr` and the `show parameters` command. Note: it is best to save the output to a file with the `spool <filename>` command and then check the details from the file. For example:

```
oracle@/u01> svrmgr
Oracle Server Manager Release 3.1.5.0.0 - Production
Oracle8i Enterprise Edition Release 8.1.5.0.0 - Production
SVRMGR> connect internal
Connected.
SVRMGR> spool para.txt
SVRMGR> show parameters
...
SVRMGR> spool off
```

  The output will be written to the para.txt file in the current directory.

- The other option is running the following SQL statements from SQLPLUS:

- ```
  SQL> spool para.txt
  SQL> select name, value
  from v$parameter where isdefault = 'FALSE'
  order by name;
  SQL> spool off
  ```

- Check the Oracle alert logs. These are found in the `$ORACLE_HOME/rdbms/log`.

  See Chapter 10, “Monitoring an RDBMS system for performance” on page 203 for more information. Oracle has a set of tools to aid in analyzing details. See 10.1.2, “Oracle monitoring tools” on page 215 for more information.

- Next, check for the obvious mistakes that ruin performance. See 13.4, “Check the most common Oracle mistakes” on page 299 for more information. Also, see 13.3, “Check the most common AIX configuration mistakes” on page 295 the for more information.
- The top ten Oracle parameters are checked next, as these need to be right before further tuning is possible. See 13.6, “Evaluate the top 10 Oracle parameters” on page 303 and 13.7, “Other key Oracle parameters” on page 310 for more information
  ```

- Then, enter the normal round of fine tuning the database and system for maximum performance. This involves tuning in the following order:
  - Access method
  - Memory
  - Disk and Disk I/O
  - CPU
  - Contention

  See 13.8, “Iterative fine tuning steps” on page 311 for more information.
- Finally, as System Administrators or Database Administrators, we rarely have control of the SQL being used on the system, but we can identify the worst offenders and highlight them to be fixed by others as soon as possible.

This approach is used in the remaining part of the chapter.

13.3 Check the most common AIX configuration mistakes

This section contains the common AIX and system administration items that were found to be wrong on database systems. These are worth checking before engaging in fine, detailed-level database tuning.

13.3.1 Change control

If the machine is important, then it should have some form of change control. To test this process, ask the following questions:

- Has anything in the system been changed recently?
- Has anything changed in the database recently?

If the answer to both is no, do not believe the answer, and, therefore, the next question must be:

- When was the last change, and what were the details?

If there is no system or database log being maintained to record changes to the system, then please start one now. Create a simple binder of pages with columns labeled:

- Date
- Who
- What changed

- Why

This would cover 90 percent of what any more complex change control system can achieve.

13.3.2 Failure to use asynchronous I/O

Asynchronous I/O is normal operation for AIX. It reduces CPU use with no risk and is recommended by IBM and Oracle.

- For details how to start AIX asynchronous I/O, see 13.9.1, “AIX asynchronous I/O” on page 318
- For how to get Oracle to use asynchronous I/O, see 13.6.3, “use_async_io or disk_asynch_io” on page 306.

13.3.3 Poor disk subsystem installation

This includes not making good use of the available hardware or configuring too many disks per adapter and forgetting availability in disk setup.

For example:

- Not balancing disks and I/O across SSA loops and adapters
- Not having mirrors on different disks and adapters

This subject is beyond the scope of this redbook, but we recommend disk configurations in:

- Chapter 6, “Sizing a database system” on page 101
- Chapter 8, “Designing a disk subsystem” on page 149

13.3.4 Redo log disks

For a system with more than a few disks, the redo log should be separated out on to a dedicated and mirrored pair of disks. For more information see:

- 13.11.10, “Oracle redo log should have a dedicated disk” on page 337
- 13.11.11, “Mirror the redo log or use RAID 5 fast-write cache option” on page 337

This is included in the AIX section because the AIX system administrator has to set this up for DBA use.

13.3.5 Paging space and monitoring paging

This is simple to forget especially on a newly set up machine:

- For paging space see, 13.9.8, “AIX paging space” on page 324
- For paging rate, see 13.9.9, “AIX paging rate” on page 324

13.3.6 Not allocating enough memory to Oracle

Oracle must have a significant amount of memory to operate at high performance levels. Although it is an Oracle tuning feature, it is included here because it is generally the AIX administrator who is best able to decide how much memory can be dedicated to Oracle use.

The initial guess will depend on the number of users, their process size, and the application. See 6.5, “Memory goals and sizing” on page 108.

Once running, any free memory should be allocated to Oracle. The AIX administrator can make careful use of the `rmss` command to determine the amount of free memory. Because AIX will make use of all memory after running for a while, we use the term *free* for memory that AIX is using, but allocating it for another purpose would not effect performance. To check for free memory, use the `rmss` command to take away two percent of memory and then releasing it. Watching the speed at which this is allocated indicates if the memory is really needed or not. See A.18, “rmss - Reduced Memory System Simulator” on page 363. Also, if paging rate is excessive, then the memory dedicated to Oracle should be reduced. See 13.11.6, “Oracle SGA size” on page 334 for more details on resizing the Oracle memory.

13.3.7 Poor use of AIX disk features

AIX has many advanced features for maximum performance and minimum workload for the System Administrator (SA) and Oracle DBA. If the database physical layout designer (this can be the SA or DBA) is:

- New to AIX
- Unfamiliar to these advanced AIX features
- Uses standard procedures that ignore these features

then the benefits of these advanced AIX features are not implemented.

This results in a database physical design using the alternative Oracle techniques to attempt to spread data across disks. These techniques assume an old and simple UNIX version and, therefore, gets Oracle to provide features now found in the AIX operating system. For example, it assumes files can only be placed on one disk. This forces the designer to place one or two large files per disk and then get Oracle to balance data across these files.

Unfortunately, if the design is not perfect, or the data in the database changes, this approach to managing disks has three implications:

- The data and disk I/O will not be spread evenly across disks, which results in hot disks and poor performance. Alternatively, AIX features will automatically spread disk I/O and avoid hot disks.
- The DBA has to constantly monitor for hot disk problems and plan for changes. If AIX features are used, monitoring and planning efforts can be reduced.
- The DBA then has to manually move tables or data files between disks to reduce hot disk problems. Often this has to be done at night with no users online. Employing AIX features means the DBA resources can be better used on other activities.

Using the advanced AIX features avoids disk performance issues (by stopping hot disks) and drastically reduces the workload of the DBA.

The traditional Oracle approach results in many sites not using these advance AIX features on their systems and includes:

- Failure to use AIX striping will reduce disk management effort and automatically reduce hot disks. See 13.9.2, “AIX Logical Volume Manager or Oracle files” on page 318.
- Failure to creating logical volumes that are easy to manage and allocate.
  - For LV sizes, see 13.9.3, “Create logical volumes at a standardized size” on page 320.
  - For a comparison of JFS and raw device files, see 13.9.4, “AIX JFS or raw devices” on page 320.
  - For making the hot files faster, see 13.9.5, “AIX disk geometry considerations” on page 322.
  - For simple administration, see 13.9.6, “Naming convention” on page 323.

13.3.8 Busy disks

There is always going to be some disks used more than others. These busiest disks are called hot disks because of the idea that a disk might heat up if it is used repeatedly due to the head movement.

The AIX administrator should always know which are the warm or hot disk disks and be ready to fix this if it becomes critical.

See 13.9.10, “Hot disk removal” on page 325 for more information.

13.4 Check the most common Oracle mistakes

This section is to remind you of the common Oracle mistakes found on systems with a performance problem. Check these on your system before going into further time consuming fine tuning details.

13.4.1 Indexes

Accidental removal of indexes and indexes being disabled, that the DBA has not noticed, are a common problem. For example: an index can get disabled when the SQL*Loader is used to directly load data, and the re-index failed. So, check the indexes for the following:

- Do all the indexes actually exist?
- Are the indexes valid (up to date and usable by the RDBMS)?
- Does the index have the right columns?
- Are all primary keys indexed (do not include trivially small tables)?

If an index is missing, there is nothing within the database to check on that. To detect a missing index, there must be a definitive list of required indexes, and the columns that should be in the index.

If there is an index problem, then fix it before tuning anything.

If indexes are missing, then use the Oracle parallel index creation feature to make the index in the shortest possible time. This works very well on SMP machines, but you may want to restart the database with larger than normal SORT_AREA_SIZE to allow fast sorting of the resulting index.

13.4.2 Analysis

Oracle depends on data about the tables and indexes. Without this, the optimizer has to guess. It is worth checking the optimizer has the following information:

- Have all tables been analyzed?
- Have all indexes been analyzed?
- Have they been analyzed after any recent changes to tables size or table structure?

If there is an analyzing problem, then fix it before tuning anything. See 13.4.4, “Analyze database tables and indexes” on page 300 for more information.

13.4.3 Basic Oracle parameters

There are a limited number of all the Oracle parameters that have a large impact on performance. Without these being set correctly, Oracle cannot operate properly nor give good performance. These need to be checked before further fine tuning.

This is covered in detail in 13.6, “Evaluate the top 10 Oracle parameters” on page 303.

It is worth tuning the database further only if all these top ten parameters are okay.

13.4.4 Analyze database tables and indexes

Oracle has warned all customers that *rule based* optimization will be dropped in future releases. As the *cost based* optimizer is now likely to give the best performance in most cases, this should be used. The cost based optimizer needs data to decide the access plan, and this data is generated by the `analyze` command.

Most parallel SQL statements, SQL hints, and many of the new performance features of Oracle, such as hash, star joins, and partitions, will only be available using the cost based optimizer. If the `analyze` command is not run, and the SQL does not use *SQL hints*, the optimizer has to use rule based optimization and will not make the new performance feature available.

The optimization mode is set in the Oracle parameters via the init.ora file with the optimizer_mode variable. Possible values are:

- CHOOSE - This means using the cost based optimizer.
- RULE - Use the rule based optimizer
- ALL_ROWS - This means the same as CHOOSE, but try to finish the query as soon as possible and maximize throughput (good for large batch queries).
- FIRST_ROWS - This means the same as CHOOSE, but try to supply the first row of the results as early as possible (good for small indexed queries).

It is usually set to CHOOSE or RULE.

It can also be set at the session level. Note that, for Oracle versions before 8.1.5, the set option is called OPTIMIZER_GOAL. From version 8.1.5 onwards, it is called OPTIMIZER_MODE like the init.ora parameter. We do not generally recommend setting this at the session level, but the syntax is:

```
ALTER SESSION SET OPTIMIZER_GOAL = RULE or
ALTER SESSION SET OPTIMIZER_MODE = RULE -- Oracle 8.1.5 onwards
```

Setting the optimizer mode to CHOOSE in the init.ora file so that the cost based optimizer is used is highly recommended in most cases. The main exception to using the cost based optimizer is when an application has been manually tuned by developers for the rule based optimizer. Even this should be tested with the latest release of Oracle to check if the cost based optimizer can now improve on the older rule based query plans and performance.

To determine if a table is analyzed, check the AVG_ROW_LEN column of the USER_TABLES table. If it is non-zero, the `analyze` command has been run at least once on this table.

To determine if an index is analyzed, check the COLUMN_LENGTH column of the USER_IND_COLUMNS table. If it is non-zero, the `analyze` command has been run at least once on this index.

If you analyze a table, then its current indexes are automatically analyzed too.

If you analyze a index, then the table is **not** analyzed.

For tables and indexes where the data is highly skewed, it is worth creating *histogram* statistics. The database usually assumes there is an even spread of values between the highest and lowest value found in a column. If this is not true, the data is skewed. For example, in England, there are many people with surnames of Smith and Jones and hardly any starting with the letter Z. This means a surname column has skewed values. Another example might be a column containing the year of the sales order. If a ten year old company is growing rapidly, it might find that the last year includes 60 percent of sales orders - this is highly skewed. To create a histogram to document the history, for example:

```
ANALYZE TABLE orders COMPUTE STATISTICS FOR COLUMN release_date
```

The default number of the histogram buckets is 75. For table columns with a large ranges and large numbers of clustered values, having a higher number of buckets can be useful.

To investigate the histograms on the system, use the USER_HISTOGRAMS table. For example:

```
SELECT * FROM USER_HISTOGRAMS;
```

Collecting the full statistics via the `analyze table <name> compute statistics` on large tables takes a lot of time and space in the system - roughly the cost

of a full table scan and sort operation. For tables with an even distribution of data in the columns, this will yield little extra value over an estimated analyze using a sample of the rows. This is performed with `analyze table <name> estimate statistics`. We recommend an estimate of 5 percent of the rows as a minimum for large tables and their indexes. For example:

```
ANALYZE TABLE orders ESTMATE STATISTICS SAMPLE 5 PERCENT
```

With the cost based optimizer, there is a further choice to make about the way the database is requested to provide the results. If your application can make use of the first few rows of the SQL statement, for example, displaying them on the users screen, then the OPTIMIZER_GOAL of FIRST_ROWS can make the application look faster to the user. This is set at the session level with:

```
ALTER SESSION SET OPTIMIZER_GOAL = FIRST_ROWS
```

In general, it is hard to code an application this way; so, it is not common. Otherwise, the OPTIMIZER_GOAL of ALL_ROWS will finish the query in the shortest possible time. This changes the access method the optimizer will choose, as some techniques provide the first rows earlier than others. The default is to maximize throughput of the system. We recommend using the default unless early screen updates are coded into the application.

13.5 Tuning hint categories for AIX and Oracle used in this chapter

The following sections contain tuning hints, and it is assumed you are a Database Administrator (DBA) or AIX System Administrator (SA).

We have categorized these tuning hints in the following way to highlight the performance difference you might expect and the risks involved.

Performance impact or benefit:

low	A 5 percent or less improvement
medium	Between 5 percent to 25 percent improvement
high	25 percent or more improvement
very high	More than 100 percent improvement might be possible

Performance risk - This is to cover the fact that some hints must be used with care, while others are only going to improve things and should always be used:

none	This is not expected to cause any problems.
low	Safe change to make.

medium You need to check this actually improves performance.

high This can cause problems or reduce performance.

The hints in the following parts of this chapter are divided into the following groups:

- Evaluate the top ten Oracle parameters
- Other key Oracle parameters
- The iterative fine tuning steps
- AIX hints
- Advanced AIX hints
- Oracle hints
- Other tuning hints (for network and compiling)

13.6 Evaluate the top 10 Oracle parameters

The non-default Oracle parameter values are worth investigating. Of course, many of them have to be non-default for the system to run at all. See 10.1.2.4, “The UTLBSTAT/UTLESTAT monitoring tool” on page 217. The parameters that make the biggest difference (and should, therefore, be investigated in this order) are listed in the following sections.

13.6.1 db_block_size

This cannot be changed after the database has been created. It is vital to get this correct before you start. As AIX does all I/O at a minimum of 4 KB, we do not recommend any sizes that are smaller than this, as it will be slower. There are reasons for using different block sizes:

- Smaller sizes
  - If the SQL statement needs just one row in a block, it is better to read a small block. If the blocks are larger, then unwanted data is read from disk, and this takes up additional memory.
- Larger blocks
  - If the typical SQL statement results in a full table scan, then larger blocks will read in more rows per disk I/O. This means more data is available for processing by the RDBMS. Note: An 8 KB read only takes a few percent longer than a 4 KB read from disk.
  - A row has to fit inside a single data block. This means the end of each block has some wasted space (because a row will not fit in exactly).

With larger blocks, there are less blocks and, therefore, less wasted space, for example, if the rows are 250 bytes and the last 200 bytes of a block cannot be used. With 4 KB blocks, there is 4.9 percent wastage, and with 32 KB blocks, only 0.6 percent wastage.

We recommend for the block size:

- For OLTP type workloads - 4 KB.
- For DSS workloads - 8 KB. Commonly, sites use 4 KB, 8 KB, or 16 KB.
- For very large databases (200 GB or more) - 16 KB.
- For mixed workloads - 4 KB.

For more details, see 13.11.5, “Oracle block size” on page 334

Performance benefit is medium and no risk.

13.6.2 db_block_buffers

This value is the number of disk blocks stored in the SGA. This is the largest part of the SGA. The memory size allocated will be:

db_block_buffers * db_block_size

If you have no idea how large to set the parameter (which is typical on a new system until you start running tests), then set it so that the SGA is roughly 40 percent to 50 percent of memory.

If your database data is in a Journaled File system (JFS), then you will need to allocate space in real memory for the AIX buffer cache. The AIX buffer cache is dynamically controlled, but you should make sure sufficient memory is available for it. However, if your data is held on raw devices, then disk I/O does not use the AIX buffer cache, and, therefore, it does not need to be large, and more memory can be allocated to the Oracle buffers. Figure 43 shows this difference.

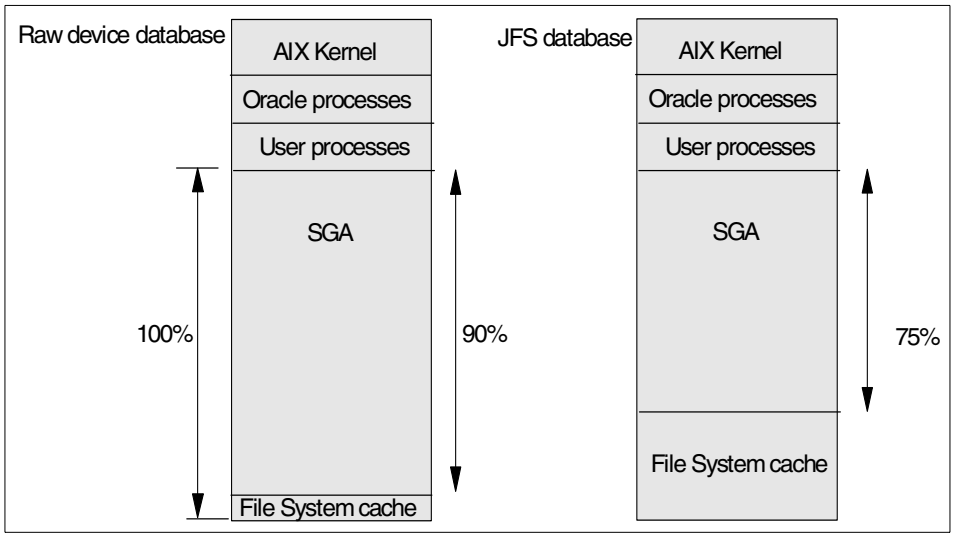


Figure 43. Different SGA buffer cache memory sizes for raw device and JFS databases

If you can estimate how much memory is left after allowing for

- AIX (32MB)
- The basic Oracle process
- The Oracle Servers
- The user application processes

then we recommend to allocate the following amount of the remaining memory:

- For JFS based databases, 75 percent
- For raw devices, 90 percent

Once the system is running, make sure it is not paging (in this case reduce the SGA size) or if there is free memory (make the SGA larger). The aim with the number of buffers is to have a very high buffer cache hit ratio (greater than 95 percent).

Performance benefit is very high, and there is no risk.

In Oracle 8, there are two more parameters that allocate space from within this set of buffers. These parameters are:

- BUFFER_POOL_KEEP - Blocks in this pool are not flushed out but kept indefinitely. This is used for tables that are vital to be cached in memory at

all times. For example, small look up tables of tables that are frequently accessed.

- BUFFER_POOL_RECYCLE - Blocks in this pool are instantly flushed, and the space is reused. This is used for tables that would rarely benefit from caching and only waste CPU resources by searching for cached blocks. For example, this is useful in DSS workloads where full table scans of extremely large tables would flush more useful data out of the cache and also when rarely accessed tables are used.

The defaults for these parameters are zero.

To get tables allocated to these two pools (rather than the standard pool) requires the DBA to use the new storage option for the table. For example:

```
ALTER TABLE EMP STORAGE (BUFFER_POOL KEEP)
ALTER TABLE EMP STORAGE (BUFFER_POOL RECYCLE)
```

The time during the preparation of this redbook did not allow further investigation of these new Oracle 8 features. Therefore, we recommend only using these extra pool parameters:

- Once the system is fully tuned using the top 10 Oracle parameters.
- If you fully understand the use of particular tables and can decide that the standard pool would not be as efficient.
- You have a test system to experiment with these pool parameters.
- You have read and understood the full Oracle documentation on these new features.

Otherwise, tune using only the db_block_buffers parameter.

13.6.3 use_async_io or disk_asynch_io

AIX fully supports the asynchronous I/O for both JFS and raw devices. Many sites do not know this and fail to use this feature.

On Oracle 7, this parameter is called use_async_io.
On Oracle 8, this parameter was renamed to disk_asynch_io, and the default is set to TRUE.

Set this parameter to TRUE, and magically your database will work faster, but note that the AIX asynchronous I/O feature must also be configured. See 13.9.1, “AIX asynchronous I/O” on page 318.

Performance benefit is very high and no risk.

13.6.4 db_writers, db_writer_processes and dbwr_io_slaves

In Oracle 7, the name of the parameter is db_writers.

This parameter decides how many database writer processes are used to update the database disks when disk block buffers in the SGA are modified. Multiple database writers are often used to get around the lack of asynchronous I/O in some operating systems, although it still works with operating systems that fully support asynchronous I/O, such as AIX.

We have recommended, in the previous section, that asynchronous I/O is used; so, we recommend you use a single database writer to keep things simple.

Therefore, set this parameter to 1 to reduce the database writer overhead.

In Oracle 8, this functionality is covered by two parameters called db_writer_processes and dbwr_io_slaves.

The db_writer_processes parameter specifies the initial number of database writer processes for an instance. In most cases, one database writer process is enough, but it is recommended to use multiple processes to improve performance for a system that writes a lot of data. The dbwr_io_slaves parameter is similar to the Oracle 7 db_writers parameters. However, if the dbwr_io_slaves parameter is used, then only one database writer will be used regardless of the setting for db_writer_processes.

Therefore, set the db_writer_processes parameter to 1 to reduce the database writer overhead and leaving dbwr_io_slaves at the default value.

Performance benefit is medium and no risk.

13.6.5 shared_pool_size

This parameter is very hard to determine before statistics are gathered about the actual use of the shared pool. The shared pool includes the data dictionary cache (the tables about the tables and indexes), the library cache (the SQL statements and execution plans), and also the session data if the multi-threaded server (MTS) is used. Its size can vary from a few MBs to very large like 100 MB, depending on the applications' use of SQL statements. Many application vendors will give guidelines on the minimum size. It depends mostly on the number of tables in the databases - the data dictionary will be larger for a lot of tables and on the number of the different SQL statements that are active or used regularly.

If you have no information, start using the following:

- For smaller systems (128 MB to 512 MB of memory) - shared_pool_size = 3MB
- For system with more than 1 GB - shared_pool_size = 30 MB

Note: Some applications that make heavy use of the this area have a shared_pool_size of up to 200 MB.

Performance benefit is medium and needs tuning based on measured statistics (see 13.11.17, “Oracle shared pool size” on page 339).

13.6.6 sort_area_size

This parameter sets the size of memory used to do in-memory sorts. In OLTP environments, sorting is not common or does not involve large numbers of rows. In Batch and DSS workloads, this is a major task on the system, and larger in-memory sorts areas are needed. Unlike the other parameters, this space is allocated in the user process and not just once in the SGA. This means that if 100 processes start a sort, then there is 100 times sort_area_size space allocated in memory; so, you need to be careful, or you will run out of memory and start paging.

Set sort_area_size to be 200 KB on a small system with a lot of users, and at 2 MB on larger systems.

Performance benefit is medium and needs tuning based on measured statistics.

13.6.7 sql_trace

This parameter makes Oracle collect information about performance. This creates an extra load on the system. Unless you require the information for tuning, this should be set to FALSE.

Performance benefit is medium, and there is no risk.

13.6.8 timed_statistics

This parameter makes Oracle collect timing information about response times. This creates an extra load on the system. Unless you require the information for tuning, this should be set to FALSE.

Performance benefit is medium, and there is no risk.

13.6.9 optimizer_mode

If the application provider recommends to set this to RULE, do so.

Otherwise, you should set this to CHOOSE (which is the default for newer versions of Oracle).

See 13.4.4, “Analyze database tables and indexes” on page 300 for more information.

Performance benefit is high, and the risk is low.

13.6.10 log_buffer

The redo log buffer is used to store the information to be sent to the redo log disk. This buffer speeds up the database performance by allowing transactions to record the updates to the database but not send nearly empty log records to the redo log disk. If there are many transactions added to the log buffer, faster than they can be written to disk, then the buffer can get filled up. This is very bad for performance.

We recommend a minimum of 128 KB, but many DBAs just set this to 1 MB to make it extremely unlikely to ever fill up.

Performance benefit is medium, and there is no risk.

13.6.11 rollback_segments

This is an odd Oracle parameter because it is a comma separated list of the rollback segment names. The rollback segments contain the original contents of blocks that are updated during of a transaction. There are two aspects to rollback segments:

- The number of rollback segments - Listed in this parameter.
- The size of each rollback segment defined when the rollback segment was created - Not listed in this or any other parameter.

For good OLTP workload performance, rollback segments are vital.

For DSS workloads:

- Large read only queries do not really use rollback segments, so they are unimportant.
- Updates to summary tables or updates during data loading the rollback segments are important.

A transaction has to place all the information into a single roll back. The roll backs can be shared between transactions, but if too many transactions share a roll back, there can be locking problems.

We recommend a lot of small roll backs. For systems with less that 32 active concurrent transactions at one time updating rows, create eight rollback segments. However, working out the number of transactions in advance is nearly impossible. If you have higher numbers of active transactions, then Oracle recommends one rollback segment per four transactions but do not go higher than 50 rollback segments.

Once the system is running you can monitor to roll back contention. See 13.11.19, “Number of Oracle rollback segments” on page 339.

Often a special large roll back is used for large batch type transactions that update large quantities of rows.

Performance benefit is medium, and there is no risk.

13.7 Other key Oracle parameters

If the above parameters are correct, then it is worth noting and investigating the below parameters in further detail. We do not intend to explain what all these parameters do in detail (unless they are covered in a later tuning section). These parameters are listed because they are the ones proven to make useful performance improvement out of the hundreds that are available. These are in roughly alphabetical order, but we have grouped some together.

1. db_block_lru_latches - Logical resources within Oracle
2. db_file_multiblock_read_count - Encourages read-ahead on sequential reads
3. dml_locks - Logical resources within Oracle
4. enqueue_resources - Logical resources within Oracle
5. hash_area_size - Similar to sort_area_size
6. log_archive_start - Archiver process, saves the redo log files to disk or tape
7. log_archive_dest - Where to copy the log to
8. checkpoint_process * - Forces database blocks to disk at a point in time
9. log_checkpoint_interval - When to force disk up-to-date based on activity
10. log_checkpoint_timeout - When to force disk up-to-date based on time
11. log_simultaneous_copies * - Controls redo log
12. mts_ - Many multi-threaded server options
13. log_small_entry_max_size * - Controls redo log

14.open_cursors - A limit

15.parallel_server - Parallel query option

16.parallel_max_servers - Parallel query option

17.parallel_min_servers - Parallel query option

18.processes - A limit

19.recovery_parallelism - Speeds up cache recovery

20.sessions - A limit

21.sort_area_retain_size - Reduces sort area dynamic resizing

22.timed_os_statistics - More performance information

23.transactions - A limit

24.transactions_per_rollback_segment - Limits roll back use

Note that parameters marked with an asterisk (*) have been removed in later versions of Oracle; so, check the documentation for your Oracle release.

There are 200 further Oracle parameters that could be investigated, but the majority of them are for special cases. These cases are when the RDBMS is not being used for a typical workload, and the standard parameters do not work well. If you suspect this is the true for your database, then you have to read the documentation in great detail before making changes, and it is wise to have some repeatable test to determine if the parameter improved performance. We do not advise changing from the default values unless you have a verifiable performance effect.

If you work through all of these parameters, there are a further 250 undocumented parameters (in Oracle 8.1.5).

13.8 Iterative fine tuning steps

In this section, we follow the Oracle guideline for the order of fine tuning the database. In the initial checking of the database, we should have already eliminated many possible causes of problems and poor performance. However, tuning is an iterative process; so, fine tuning, for example, the memory use might effect disk I/O. This means these have to be rechecked on each iteration.

13.8.1 Access method tuning

The first step is to understand the SQL statements running on the RDBMS. Tuning this can have the largest impact on performance once all the common

mistakes have been removed by the previous stages of tuning. Without knowing the SQL running on the system, you are tuning in the dark.

13.8.1.1 Tuning SQL

Although we have explicitly excluded the tuning of SQL statements from this redbook, this is the tuning step at which it should be attempted. The tools covered in 10.1.2, “Oracle monitoring tools” on page 215 give an overview of how to investigate SQL statements. In addition, the TKPROF facility along with EXPLAIN PLAN can help isolate the worst SQL statements in the system.

Attention

To use TKPROF, the trace options SQL_TRACE and TIMED_STATISTICS have to be enabled, and this can have a large impact on the performance of the database in terms of CPU utilization to capture the data and disk I/O to write the trace files to disk. So, remember to set these to false once tuning is complete.

Please refer to the Oracle manuals and the various books on tuning SQL for further explanations of TKPROF, EXPLAIN PLAN, and UTLBSTAT/UTLESTAT.

13.8.1.2 Transaction rates

There is a simple way to find the transaction rate of the system that seems to be hard to find in the manuals and books. The *v$sysstat* table includes the total count of commits and roll backs the database has performed since startup. So, periodically saving the counts will allow the calculation of the transaction rates. Use the SQL statement from B.2.1, “Oracle number of transactions” on page 374.

13.8.1.3 Parallelization for large query sorts and indexing

Both of these operations benefit from parallelization, provided there is space and capacity on the machine. This is particularly likely on SMP machines. Large queries can be parallelized via *SQL hints* or parallelization set on the tables. The index parallelization is set on the `create index` command, and often large indexes are created at off-peak times when the whole power of the machine is available.

Please refer to the Oracle reference material for further explanations of SQL hints and the `create index` command.

13.8.1.4 CREATE TABLE AS SELECT with unrecoverable option

This `CREATE TABLE AS SELECT` SQL statement is often referred to a *CTAS.* Oracle allows parallelization of the `SELECT` and of the `CREATE TABLE` commands. This is an excellent way to generate one table from another at high speed.

Please refer to the Oracle reference manuals for further explanations of the `CREATE TABLE AS SELECT` command.

13.8.2 Memory tuning

Once the access methods have been investigated (see previous section) the use of memory is next. This involves checking the sizes and use of memory.

13.8.2.1 Paging

Check the system is not paging.

See 13.9.9, “AIX paging rate” on page 324 for more details.

13.8.2.2 SGA db_block_buffers

This Oracle parameter decides the size of the largest part of the Oracle SGA. It is the most important parameter for Oracle performance.

See 13.11.6, “Oracle SGA size” on page 334 for SGA size and 13.11.8, “Oracle buffer cache hit ratio tuning” on page 336 for setting this parameter. There is also a discussion about this in 6.5.3, “RDBMS cache and structures” on page 109.

13.8.2.3 SGA redo log buffers

Check if the size of the redo log buffer is sufficient by checking the redo log space request row of the v$sysstat table.

See 13.11.16, “Oracle redo buffer size” on page 338

13.8.2.4 SGA shared pool

This is really an internal scratch pad area of Oracle, but it can cause problems. If not sufficiently large, this will cause poor performance. If far too large, it wastes space that the buffer cache could use.

See 13.11.17, “Oracle shared pool size” on page 339 for more details.

The dictionary cache and library cache are parts of the shared pool.

13.8.3 Disk I/O tuning

Use the standard AIX tools, such as `iostat`, to monitor disk activity or advanced tools, such as `filemon` or the graphical Performance Toolbox/6000 for tuning data. Alternatively, use one of the unsupported but useful tools, such as `nmon` (see A.15, “nmon - online monitor” on page 361).

13.8.3.1 Redo log

In systems with more than a handful of disks plus high insert and update rates to the database (which means most databases), it is recommended to dedicate a disk for the redo log.

See 13.11.10, “Oracle redo log should have a dedicated disk” on page 337 for more details.

13.8.3.2 Balanced disks and hot Oracle files

Disks are the biggest area of tuning because there are a lot of options. If you have followed the advice of this redbook, you should have avoided the majority of performance problems with disks. The following are areas to check and correct once the system is running with a real workload.

- Which method of disk balancing is in use - See 13.9.2, “AIX Logical Volume Manager or Oracle files” on page 318.
- If a hot disk is detected, how to reduce the impact - See 13.9.10, “Hot disk removal” on page 325.
- To never have one hot disk - See 13.9.11, “Disk sets for hot disk avoidance” on page 325.
- To see Oracle’s view of hot files in the system, see the output from UTLBSTAT and UTLESTAT - See 10.1.2.4, “The UTLBSTAT/UTLESTAT monitoring tool” on page 217.

13.8.3.3 RAID 5

If it turns out that the there is a lot more write to disk than expected, RAID 5 performance can be an issue if the fast-write cache option is not used.

13.8.3.4 Double check asynchronous I/O

It is worth double checking the asynchronous I/O is in use.

See 13.9.1, “AIX asynchronous I/O” on page 318 and Part 13.11.7, “Oracle database writers” on page 335.

13.8.3.5 Fragmentation on extents and tablespaces

As databases get bigger, the extents, tablespaces, and files get bigger. The extent fragmentation problem has become much less important.

For a detailed output of the extents in the database, use the sample script in B.2.12, “Oracle report on extents” on page 378.

13.8.3.6 Use raw devices

If the database is JFS based, and there are still disk I/O issues, it is worth considering moving to raw devices.

See 13.9.4, “AIX JFS or raw devices” on page 320.

13.8.4 CPU tuning

Next, there are some things that can be checked and tried to reduce CPU problems. Also, note that reducing disk I/O saves CPU power because the disk device drivers are called less - this is why CPU tuning is after disk tuning.

13.8.4.1 Balanced SMP

Check that all the CPUs on an SMP machines are actually being used.

See 13.9.12, “SMP balanced CPU utilization” on page 326 for more information.

13.8.4.2 Parallelism

Over use of parallelization can cause large CPU problems.

See 13.11.20, “Oracle parallelization” on page 340 for more details.

13.8.4.3 Time slice

Altering the time slice of the machine can help by reducing the CPU cycles required to reschedule processes.

See 13.10.8, “AIX process time slice” on page 329 for more information.

13.8.4.4 Balance the users

One approach that is often missed out is trying to organize the users and tasks of the system better. Ask these questions:

- Can the users spread out their workload during the day to avoid high peaks in computer workload?
- Can they request reports overnight or at low priority?
- Can reports be generated from a queue to stop them flooding the system?

- Are there better ways to look up customer accounts and avoid full table searches?

These question require that user behavior is observed and careful suggestions are made. Surprisingly, many users are willing to participate, give ideas, and change work patterns if this might help them to avoid waiting for the computer and make their working day more pleasant.

13.8.4.5 Processor affinity

On an SMP machine, the processes are scheduled to run on the next available CPU. If possible, they run on the same CPU, as this increases performance because the CPU level 1 and level 2 caches already contain code and data for that process. This results in less calls to load from main memory. Improved scheduling algorithms mean the chances of running on the same CPU are increased. The alternative is forcing the process to always run on a particular CPU.

See 13.10.5, “AIX processor binding on SMP” on page 328 for more information.

13.8.4.6 Move the application

Oracle is fully client server enabled; so, to reduce the CPU used on the database server some applications can be moved to other, existing systems, or a new machine can be used. This can be effective if:

- The application is implemented on AIX (rather than directly on the PC).
- The PC version of the application can be used instead.
- The application is a batch process that can be moved to a new machine.

Moving to client/server may mean slower performance for the application, but the database server should run faster.

13.8.5 Contention tuning

Contention is the result of processes fighting for resources. This can simply be called an internal Oracle problem. Fortunately, there are Oracle tuning options to allow this to be monitored and controlled.

Contention is hard to spot on a system but is typically highlighted by poor performance when the machine does not really look or feel particularly busy. This means that tasks are being serialized rather than being concurrent, or processes are waiting for each other and relying on time-outs to resolve the problem.

The `utlbstat` and `utlestat` scripts report the important information from the v$system_event table (see 10.1.2.4, “The UTLBSTAT/UTLESTAT monitoring tool” on page 217 for more information). See the Oracle manuals and Oracle 8 Server Tuning for more details. A few of the important contention areas are covered in the following sections.

13.8.5.1 Roll back contention

Roll backs contain the before images of rows while a transaction is running. There can be problems with these roll backs when too many processes try to access them.

See 13.11.19, “Number of Oracle rollback segments” on page 339 for more information.

13.8.5.2 Redo log buffer latch contention

The redo log buffer holds the information to be written out to the redo log disk, but if many transactions are trying to add details at the same time, this can cause a bottleneck.

See 13.11.15, “Oracle redo buffer latch” on page 338 for more information.

13.8.5.3 Parallel query server contention

If the database is using the parallel query option, then it is possible to either run out of parallel query servers or overdo the parallelization and try to run too many processes.

See 13.11.20, “Oracle parallelization” on page 340 for more details.

13.8.5.4 Spin count

When Oracle has to wait for a resource, it can either go to sleep and be restarted when the resource is free, or it can just keep retrying repeatedly, but this takes up CPU time. Retrying only makes sense on SMP machines and assumes the resource is locked for very short periods of time and it is worth waiting. The spin count is the number of retries before sleeping.

See 13.10.6, “AIX spin count on SMP” on page 328 for more information.

13.9 Tuning AIX for Oracle hints

This section contains the most common AIX tuning hints that should be considered as normal operation. They are not in any particular order.

13.10, “Advanced AIX tuning hints” on page 326 includes advanced AIX hints that must be used with caution.

13.9.1 AIX asynchronous I/O

In the Oracle init.ora configuration file, set this to TRUE. For Oracle 7:

```
use_async_io = true
```

and for Oracle 8:

```
disk_asynch_io = true
```

Then set the *minservers* and *maxservers* using the AIX `smit` command SMIT->Devices->Asynchronous I/O->Change/Show Characteristics of Asynchronous I/O (or just type `smit aio`) to:

- MaxServers = 10 * number of disks but with a maximum of 10 times the number of processors in the machine.
- MinServers = MaxServers /2

For example, on a machine with 30 disks, then Max Servers = 300 and MinServers should be 150, but if this machine only has four CPUs, the MaxServers = 40 and MinServers = 20 is sufficient. Higher numbers will not hurt performance, as it only results in more kernel processes running that do not actually get used.

Using asynchronous I/O is likely to increase performance. There is no risk of reducing performance; so, it should always be used.

Performance benefit is high and no risk.

13.9.2 AIX Logical Volume Manager or Oracle files

To spread out the data and disk workload across disks, you have two choices:

1. **Use Oracle** - As the Oracle DBA, use a lot of Oracle files and place them onto disk by creating a logical volume on a particular disk, then monitor the database objects (tables and indexes) in the files and move objects between files to balance disk use. In this case, the AIX System Administrator simply creates each Oracle file (JFS or raw logical volume) on a separate disk.
2. **Use AIX** - As the AIX system administrator, use the AIX Logical Volume Manager (LVM) to create each JFS or raw logical volume on multiple disks. Use striping to spread data across disks and then monitor the AIX physical volumes. In this case, the Oracle DBA does not have to balance disk use between Oracle files and, therefore, disks.

These two options might sound similar, but there is one main point. Either AIX spreads the disk work out across the disks, or the Oracle DBA has to do it. That is, either a human does it, or the computer does it automatically. Sites and people familiar with Oracle tend to avoid using AIX LVM. This is a mistake.

It is strongly recommended by IBM and Oracle that the benefits of the AIX LVM are fully used.

The AIX LVM has a number of options, and striping data across disks is very effective, as it makes full use of the disks in usage terms, makes excellent use of read-ahead for sequential I/O, and spreads disk I/O evenly for better performance. For striping, use the following:

- Stripe unit size = 32 KB or 64 KB
- max_coalesce = 64 KB
- minpgahead = 2
- maxpgahead = 16

For a full explanation of these parameters, see the AIX documentation or the redbook *RS/6000 Performance Tools in Focus*, SG24-4989.

Note:

- The striped LV size must be a multiple number of the drives used. For example, if the strip size is 32 KB, and the LV is spread across eight disks, then the size of the LV must be a multiple of 32 * 8 K. This allows the LVM to create the LV with a complete number of stripes and avoid the case that the last stripe does not cover all the disks.
- It is recommended that striped data and the database logs are on different sets of disks.
- Before AIX Version 4.3.3, striped logical volumes could not be mirrored. This resulted in striping not being used or sites opting for RAID configurations. There is an alternative that gives the load balancing effect of striped and mirrored disks that can still be used, called PP level striping, that works for large files (large being over 100 MB). The logical volume is created on a set of disks with the INTER-POLICY set to maximum. This results in the Physical Partitions (typically 4 or 8 MB chunks) being spread across the disks one after the other.
- After AIX 4.3.3, the LVM does allow striping and mirroring at the same time. This has been tested for performance and works well.

Benchmarks have shown using AIX LVM to stripe data can increase performance on disk bound systems by as much as a factor of three. Disk bound batch workloads particularly benefit from this. Using the LVM to spread and stripe data across disks reduces hot disks and reduces DBA man-power in monitoring and moving data within the database.

Performance benefit is high and no risk.

13.9.3 Create logical volumes at a standardized size

When creating logical volumes in order to use them as Oracle files, create all of them with standard size. You might decide on two or three standard sizes, but make sure they are multiples of each other and will fit nicely into the standard disk sizes on the system.

For example, a standard 4.5 GB disk drive, when assigned to a volume group with a Physical Partition (PP) of 8 MB, contains 537 PPs, which means 4296 MB of space for logical volumes. This would make four logical volumes of 134 PPs each, which is 1072 MB. These disks are then grouped together for striping and mirroring. In practice, you might like to use a binary number, such as 128 PPs and 1 GB files.

We recommend making all the files 1 GB or 2 GB is size on large machines. This size of file does not cause large file problems and can be copied or moved around the system. It is worth staying under the 2 GB limit because larger files can still cause problems and lower performance. Some applications, tools, and commands have problems with files larger than 2 GB because they have not been re-coded to make use of the 64 bit file system calls required to handle larger file sizes. To hold files larger than 2 GB in a JFS requires the use of the *large file support* version of the JFS. This is not as efficient as the regular JFS. There are occasions when smaller files are needed in the database, and we recommend using 64 MB as a minimum.

These sizes will mean the number of files is kept down to a reasonable limit and are still usable.

On small machines, try using 256 MB files.

Performance benefit is medium, and there is no risk.

13.9.4 AIX JFS or raw devices

This is a much discussed subject with arguments in favor of both sides. In the books and manuals, there are almost religious opinions that are completely opposite. The two options are:

- JFS - If your database is not I/O bound, that is, your applications do a lot of computation on small or infrequently retrieved data, then the JFS is a good choice because its simpler to create, work with, administer, and back up/recover.

  Reading a block from a JFS file means it is moved from disk to the AIX buffer cache and is then copied from the AIX buffer cache to the Oracle SGA for the process to access the data.

  Because there is a copy in the AIX buffer cache, disk I/O may be avoided in the case where the SGA copy was removed (reused for other purposes), but the AIX buffer cache copy survived.

  Note that when Oracle writes data to the database, in most cases, it has to force the data to disk, which results in a copy from SGA to AIX buffer cache and then moving the block out to the actual disk drive.

  The extra overhead is incurred because the block is in the JFS cache (taking up memory), and AIX has to lock the data structures and maintain the inode details and indirect block structures (taking more CPU cycles).

- Raw devices (also called raw logical volumes, raw partitions, or raw disks) are harder to work with, as they do not appear in the file system and, for example, the files cannot simply be copied with the `cp` command. They are harder to back up and recover, as the `dd` command has to be used. Raw devices avoid the double buffering of data. This means the data is read from disk straight to the Oracle SGA. The same is true for writing to a raw device. Because the database does not use the AIX buffer cache for data access, the AIX buffer cache size can be a lot smaller, and, thus, memory is freed up and can be used to support a much larger Oracle SGA. For example, memory with a JFS based database might be:

  - 33 percent for processes
  - 33 percent for SGA
  - 33 percent for AIX buffer cache

  For raw device databases, this might be:

  - 33 percent for processes
  - 60 percent for SGA
  - six percent for AIX buffer cache

  But, and it is a big BUT, raw devices are faster (see 8.10, “Disk performance measurements and observations” on page 178).

The number of databases that cannot benefit from faster disk access is small. Therefore, raw devices are recommended for performance reasons.

The main problem is that some backup systems do not directly support the backing up of raw devices. Before moving to raw devices, check that you can back the files up to tape or disk using your backup method.

From benchmark experience, moving to raw disks for disk I/O bound systems is likely to increase performance by 0 - 50 percent provided the SGA size is also adjusted to counteract that the AIX buffer cache is no longer used.

Performance benefit is high and no risk.

Note

Also the move from JFS to Raw devices is a simple process and does not require exporting and importing the whole database but can be done file by file.

13.9.5 AIX disk geometry considerations

With the AIX LVM, you can place data (logical volumes for JFS use or raw devices) on particular positions of the disk. These position on the disk are called:

- (outer) edge
- (outer) middle
- center
- inner middle
- inner edge

The center part of the disk is the fastest because, on average, the seek time to the center position is less than a seek to or from the edges of the disk, which causes higher head movements. Use the `-a` option of the `mklv` command to set this or, in smit it is the `POSITION on physical volume` option see 8.4.2, “Use of LVM policies” on page 156).

If the disk center position is the only part of the disk that is used, then the disk average seek times will be much faster. This means the disk will appear to have lower seek times than the average quoted for the entire disk. This option should be considered if you:

- Do not need the unused edge parts of the disk (you have spare disk capacity).
- Have highly performance critical parts of the RDBMS where maximum performance outweighs the extra costs.

This may increase performance by up to 10 percent.

Logical Volume placement

When creating a database, create the performance critical logical volumes first and in the center of the disks to ensure they get the best possible performance.

The performance benefit is medium and no risk.

13.9.6 Naming convention

When creating volume groups, logical volumes, and file systems, decide on a naming convention and stick to it. We recommend creating all these via scripts to make sure mistakes are unlikely. Any clear and consistent policy is better than none at all. Many sites only allow one person (or group) to create logical volumes to make sure the policy is adhered to.

Most people end the volume group name with *vg*.

For logical volumes, include something in the name to indicate the owning subsystem and the purpose, such as *ora* for Oracle and *data*, *idx*, *tmp,* or *log* for the various parts of the database. This makes performance tuning much simpler, as the logical volume name tells you what sort of data it contains, and the system characteristics are simpler to monitor.

The performance benefit is zero, but it makes tuning simpler, and there are no risks.

13.9.7 AIX sequential read ahead

This only affects JFS file system based database files.

The Virtual Memory Manager spots sequential reading of JFS based database files by watching the access pattern of read system calls. After a number of sequential reads are noticed, it will attempt to read up to the maxphahead blocks of the file in advance. By default these, are:

- minpgahead 2
- maxpgahead 8

These can be increased to increase sequential reading ahead of data and; so, speed up sequential reads using `vmtune`. For example:

```
vmtune -r 512 -R 1024
```

Keep the numbers powers of 2.

For a full explanation of these parameters, see the AIX documentation or the redbook *RS/6000 Performance Tools in Focus*, SG24-4989.

The performance benefit is medium, and risks are low.

13.9.8 AIX paging space

Never run out of paging space.

Use: `lsps -a` to determine the size and usage of the current paging space.

We also recommend spreading out paging space onto multiple disks. As a rule of thumb, only put 1 GB of paging space on any one disk. If the system starts to heavily use paging space, having 9 GB of paging on one disk means it instantly has a disk I/O problem, but by having nine times 1 GB paging area on nine disks means the temporary paging problem will be sorted out nine times faster.

The performance benefit is low, and there are no risks unless you run out of paging space, at which time, it can be very bad.

13.9.9 AIX paging rate

Paging in any UNIX is very bad news. It can very seriously reduce performance. If paging gets bad, AIX will start swapping processes out too. Ideally, paging should be completely avoided. When users start and stop large processes, there is likely to be some limited paging (this is how AIX gets the program into memory). But, this paging should be limited and short lived, and no paging for long periods is the best.

Check to see what is in memory with `ipcs` and `ps aux` (check the RSS column values) and use the `vmstat` command to monitor paging. See Appendix A, “AIX performance tools summary” on page 353 for more details or the AIX manuals.

The `vmtune` command can adjust AIX paging parameters.

Also refer to the redbook *Understanding IBM RS/6000 Performance and Sizing*, SG24-4810, for more information on monitoring paging.

In large configurations, such as the RS/6000 S Series with 12 to 24 CPUs, AIX can support thousands of users and their processes. With large numbers of programs running, there is a high chance that they are going to change their working sets. This causes regular paging. So, for this size of machine,

benchmark people use the rule 10 pages per second per CPU. So, for a 12 way machine, 120 pages per second is acceptable - zero is preferred.

The performance benefit is high, and the risks are medium.

13.9.10 Hot disk removal

First, always know where you have spare capacity for both disk space and spare disk I/O capability. When an emergency happens, you can avoid having to work this out, and the knowledge about it will save you time.

Once you have determined (use the `filemon` command, see A.2, “filemon - File I/O Monitor” on page 354 for details) which is the hot disk, you have to find out which file on that disk is causing the I/O and whether it is read or write or both. The choices are then to:

- Move that file completely to a new disk with space capacity.
- Move part of that file. AIX allows the Physical Partitions (PP) of a logical volume to be moved to alternative disks. So, either the raw device can be partially moved to alternative disks, or the JFS containing the file can be partially moved.
- Move the contents of that file (in Oracle terms, you could move the table to a different tablespace or recreate the table in alternative extents).
- Stripe the data across multiple disks to spread the I/O between disks rather than just moving the problem. Use striping and mirroring in AIX 4.3.3. Before this release, you can use PP level striping and mirroring.

The performance benefit is high, and the risks are low.

13.9.11 Disk sets for hot disk avoidance

This redbook consistently recommends disk protection and avoiding hot disks within the database.

For both RAID 5 and striped mirror disk protection, the data is spread across multiple disks:

- For RAID 5, we recommend seven disks plus one parity disk for an eight disk RAID 5.
- For striped mirror (both PP level or fine stripe on AIX 4.3.3), we recommend spreading across eight or 16 disks.

The result is that a single hot disk is impossible, but you can still have hot eight or 16 disk sets. But, the performance problems will be eight (or 16) times smaller as a result.

It is traditional, and Oracle's recommendation, to split index, data, and temporary tablespaces onto different sets of disks. This means we should have these groups of eight (or 16) disks assigned to one purpose or another.

The performance benefit is high, and the risks are low.

13.9.12 SMP balanced CPU utilization

Most large systems are now SMP machines, as this allows multiple fast processors to scale up to higher power machines. There is one small drawback in that the application and database must have enough processes to keep all the CPUs busy. If only one process is used, only one CPU can be used (assuming it is a single threaded application), and, therefore, only a fraction of the available power of the machine is used.

Fortunately, Oracle has multiple processes (one per user as a default) and for large tasks allows it to be parallelized to make it effective on an SMP.

It is worth checking that this is actually happening, particularly for batch jobs that have, in the past, often been implemented as a single process.

Use the `sar -P ALL 1 10` command to check if all CPUs are busy.

The performance benefit is high, and there is no risk.

13.10 Advanced AIX tuning hints

This section includes hints for advanced AIX options. You are normally not expected to use these unless:

- You have tried everything else.
- You fully understand what these option do.
- You are running a benchmark and need the last one percent in performance.

13.10.1 AIX readv() feature

This option is not available in Oracle 8.

Using the readv() feature of AIX can improve performance, but it can also reduce performance. The actual effect must be tested before this can be

recommended on your system. In the Oracle Version 7 init.ora configuration file, set:

```
use_readv = TRUE
```

This effectively asks the AIX kernel not to buffer reads (particularly JFS files) and should increase performance. Because of the risk of reduced performance, it is recommended only to try this feature when desperate for extra performance.

The performance benefit is low, and the risk is high.

13.10.2 AIX direct I/O

This is available in AIX but not implemented within Oracle. We put this in here because other platforms that run Oracle do use this feature, and, at least, you know that this is not an option on AIX.

13.10.3 AIX write behind

This only effects JFS file system based database files.

This effects the way AIX writes JFS files to disk. Disable the AIX feature by setting the AIX parameter using:

```
vmtune -c 0
```

Note: To set the AIX parameter back to normal, use:

```
vmtune -c 8
```

Do not use this feature unless the machine is a dedicated database server.

For a full explanation of these parameters, see the AIX documentation or the redbook *RS/6000 Performance Tools in Focus*, SG24-4989

The performance benefit is low, and the risk medium.

13.10.4 AIX disk I/O pacing

Disk I/O pacing is an AIX feature that stops disk I/O intensive applications flooding the CPU and disks. This is changed via the low and high watermarks via:

`SMIT`->**System Environment**->**Change/Show Characteristics of OS**.

If not set correctly, this can reduce performance. Test the effect of disk I/O pacing before and after changing from the default values.

The performance benefit is medium, and risk is low.

13.10.5 AIX processor binding on SMP

A process (for example, Oracle processes or the application process) can be forced to run on only one particular CPU of an SMP machine. The benefit is increased CPU cache memory hits. The down side is that the process then cannot be dispatched to an unused CPU. If the CPU to which it is allocated is busy running another process at the same priority, then the process cannot (as is normally the case) be allocated to another CPU.

Use the `bindprocessor` command.

This feature is only available on AIX Version 4 and above. You can use this feature to bind the main Oracle processes to different processors with good effect. Also, if the SQL*Net listener is bound, all the processes it creates (for example the Oracle servers used to connection the remote application process) are also bound to that CPU. This can be used to limit remote connects to particular processors.

This may increase performance by 15 percent.

Note, this is rarely done on production machines because it can also reduce performance unless carefully and permanently monitored and measured. For a full explanation of these parameters see the AIX documentation or the redbook *RS/6000 Performance Tools in Focus*, SG24-4989.

The performance benefit is medium, and the risk is medium.

Note

In AIX 4.3.3, many improvements have been made regarding processor affinity within AIX to improve performance, and also Workload Management has been introduced. Therefore, the `bindprocessor` command is hardly of any benefit any more with AIX Versions from 4.3.3. onwards.

13.10.6 AIX spin count on SMP

This parameter was removed in Oracle 8.1.5 (it is a hidden parameter, and we recommend that it is not changed). It can reduce, in earlier Oracle versions, internal latch contention. In the Oracle init.ora configuration file, the default is:

```
spin_count = 2000
```

Increasing this means the process will spin longer waiting for the process on other processors to free the latch so that it can immediately continue. Setting this to zero can help when CPU usage is very high. The latest and fastest processors can benefit from a large spin_count (because they spin faster, and, therefore, a higher count is needed to make the waiting for the latch duration to be the same time as on slower machines). In this case, doubling the spin count can increase throughput.

The performance benefit is medium, and the risk is medium.

13.10.7 AIX process priority

Attention

Getting this wrong may crash your machine.

Increasing the priority (reducing the number) can improve performance if there are a lot (hundreds) of runnable processes on the machine. Oracle Version 7 provides a `setorapri` command to do this:

```
setorapri 39
```

This may increase performance by 15 percent. This is not available for Oracle 8.

Only the root user can set this using the setpri() system call. Normally, you need to be a C programmer to use this option.

The performance benefit is medium, and the risk high. Therefore, we do not recommend to use this functionality.

13.10.8 AIX process time slice

Only the root user can set the process time slice value using the `schedtune` command. This is found in the bos.adt.samples fileset and will be /usr/samples/kernel/schedtune. This command can increase the time a process is allowed to run on a CPU before the scheduler removes it in favor of another runable process.

If it is found on a very busy system with hundreds of processes in which many Oracle or application tasks nearly finished but then get taken off the CPU and have to wait to be scheduled again to finish the task. In this case, letting the process run a little longer would mean the task is finished and, thus, reduce response times. This is not easy to determine. The best way is to increase the time slice and try to measure the effect on performance.

We do not recommend to test this on production systems. Check the AIX documentation or the redbook *RS/6000 Performance Tools in Focus*, SG24-4989, for further information.

The performance benefit is low, and the risk medium.

13.10.9 AIX free memory

For JFS database files, there can be a copy of the disk block in both the Oracle SGA buffer cache and in the AIX buffer cache. This double buffering can affect performance and cause disk I/O bottlenecks. There are two AIX buffer cache tuning parameters that determine the size of the free list:

- minfree - Below this limit, page stealing starts trying to reclaim memory pages.
- maxfree - Above this limit, page stealing stops.

The numbers are the number of pages. For a full explanation of these parameters, see the AIX documentation or the redbook *RS/6000 Performance Tools in Focus*, SG24-4989

Increase minfree and maxfree using the `vmtune` command. This way, the read page ahead algorithm does not reduce the amount of free memory pages all the way down to zero so that there is always free memory for disk I/O. AIX naturally tries to keep used memory pages for as long as possible in case the contents can be reused and; so, disk I/O can be avoided. This means the memory free list on AIX is always small (after the system has been running for a while).

On machines with large memory (512 MB or more), you should try to keep a small amount of free memory. Making minfree and maxfree larger should increase the free memory slightly. This means always wasting a little memory, but also means disk I/O is not delayed. For example, keeping 128 pages free by using:

```
vmtune -f 128 -F 144
```

On machines with less memory (less than 512 MB), do not change these parameters.

We recommend that only experienced AIX administrators change these limits, as it can, if set wrong, cause a system to perform slowly or strangely.

The performance benefit is medium, and the risk is medium.

13.10.10 AIX buffer cache size

This depends much on the workload and I/O characteristics of your database and whether you are using a JFS file system based database or raw devices.

There are two AIX buffer cache tuning parameters that determine the AIX buffer cache size:

- minperm - Below this limit, file and code pages are stolen.
- maxperm - Above this limit, only file system pages are stolen.

The numbers are pages of memory used by the buffer cache. A simplistic way of remembering this is that AIX will try to keep the AIX buffer cache size between minperm and maxperm percentage of memory. Use the `vmtune` command with no parameters to determine the current values of the minperm and maxperm. At the bottom of the output is the total pages in the system. Look for:

```
number of valid memory pages = [number]
```

Also, at the bottom of the output is the percentages of memory that the values of minperm and maxperm work out too, which is often more helpful. To change minperm and maxperm, you have to work out the actual number of pages that will work out to the percentages you are aiming for.

The defaults should work out to approximately 20 percent and 80 percent of memory.

13.10.10.1 Buffer cache size for a JFS based database

For JFS based databases, you are using the AIX buffer cache.

On machines with large memory (1 GB or more), you will find that 20 percent of memory is not available for file system cache (200 MB). This is a large amount of memory. There are two cases to consider:

- On systems that have only a few or small applications, this memory is not all used up by the application or RDBMS code. In this case, raising *maxperm* will make more of this memory available for AIX buffer cache use. For example, change maxperm to 95 percent of memory.
- On systems with very large applications, you might find that AIX keeps stealing application code pages, which results in continuous paging of application code. In this case, lowering maxperm will allow higher percentage of memory to be allocated to application code and; therefore, reduce paging. For example, change maxperm to 70 percent of memory.

On machines with less memory, do not change these parameters or be very careful, as the default values have been found to work well.

13.10.10.2 Buffer cache size for a raw device based database

For raw device (also called raw disk, partition, or logical volume) based databases, you are not using the AIX buffer cache to any great extent. It is actually being used for disk I/O for AIX processes and, for example, RDBMS error log files, but the bulk of the RDBMS disk I/O is bypassing the AIX buffer cache (that is a major point of using raw devices).

On machines with large memory (say 1 GB or more), that are only running an RDBMS, you will find that between 20 percent and 80 percent of memory has been earmarked for the AIX buffer cache. But the Oracle SGA is occupying a large part of memory. For example, the SGA might be 50 percent of memory; so, the other 50 percent is used shared between processes and buffer cache. This means the values of 20 percent to 80 percent are not a sensible setting. We might page out process memory (code or data) from memory unnecessarily.

When the system is running normally, use the `vmtune` command with no parameters to determine the amount of memory used for the buffer cache. At the end of the output, you will find a line like:

```
number of file memory pages=[number] numperm=[num] percent of real
memory
```

The second number (the *numperm* percentage) is the one to think about. If this is less than the *minperm* (on the line above), then we have the memory pages being stolen from code and file system buffer cache equally. This results, probably, in unnecessarily paging out processes.

Another way to look at this is to say that the file system buffer cache should be 20 percent to 80 percent of the memory *not* occupied by the Oracle SGA. If, for example, the SGA is 50 percent of memory, then the buffer cache should be 10 percent to 40 percent. The important value is that of minperm. You could never reach the 80 percent default value of maxperm because the SGA is taking up a large part of memory.

There are a number of cases to consider:

- If minperm is greater than 20 percent of the non-SGA memory, set minperm to be this value and reconsider once the system has settled down.
- If the numperm number is greater than minperm, you should consider allocating more memory to the Oracle SGA.

- If numperm is smaller than minperm, you are freeing processes memory and should reduce minperm. For example, change minperm to 2 percent of memory.

The performance benefit is medium, and the risks are low.

13.11 Oracle tuning hints

This section contains tuning hints for Oracle itself but specifically running on AIX.

13.11.1 Oracle installed according to Oracle Flexible Architecture

Oracle has defined a standard way to install Oracle on UNIX systems and how to layout the directory structure. This is called the Oracle Flexible Architecture (OFA). This should be followed closely. It has been developed from a great deal of experience. Many DBA tasks are simplified by using this standard, and it will help anyone joining the team to understand where everything is placed.

The performance benefit is medium, and there are no risks.

13.11.2 Oracle ARCHIVEMODE

Never ever run your database in NOARCHIVEMODE, as it cannot be recovered.

The performance benefit is none. It is a recovery issue, and there are no risks.

13.11.3 Oracle control files

Always have three control files on different disks. This might be a case for having one of them on the AIX operating system disk just to make sure you always have one available for emergencies.

Performance benefit is none as it is a recovery issue and there are no risks.

13.11.4 Oracle post-wait kernel extension for AIX

This reduces the overhead of semaphore operations for interprocess communication and locking resources internally to Oracle processes. This is mandatory to use, or Oracle will not even start. The extension is loaded as part of the AIX initialization at boot time and is loaded via the /etc/inittab file and the init process.

Make sure the correct version is installed, that is, the one supplied with your current version of Oracle and not from a older version of Oracle or the one found on the machine. This file is placed in the AIX /etc directory with the name /etc/pw-syscall (or /etc/pw-syscall4.1 with older versions of Oracle).

The performance benefit is high, and it is mandatory.

13.11.5 Oracle block size

The block size is set at `create database` time and cannot be altered at a later date without exporting the entire database, re-creating it, and reloading the database.

We recommend the following Oracle block sizes:

`db_block_size=4096` for:

- Small databases (less than 10 GB).
- JFS based databases (because AIX does 4 KB pages).
- OLTP workloads where you typically only want one row in the block and reading an extra block would not help.
- Mixed workload (OLTP and DSS) databases to assist OLTP performance.

`db_block_size=8192` for:

- Large database (greater than 10 GB)
- DSS workload where reading more rows in one go can help.
- Large rows where most of the rows of the database are large, and you will get less wastage at the end of blocks with larger blocks.
- Databases with batch workloads where the dominant database load involves using large table scans (and not indexed single row accesses).

`db_block_size=16384` for:

- For very large databases (greater than 200 GB) with DSS workloads.

The performance benefit is medium, and there are no risks.

13.11.6 Oracle SGA size

The Oracle SGA is a group of shared memory structures that contain the database buffer cache, the shared pool, the log buffer, and the data dictionary. The Oracle SGA cache size parameters (db_block_buffers and shared_pool_size) are the two most critical parameters in the Oracle system.

They set the sizes on the two important caches within the Oracle SGA. These caches make a large contribution to the RDBMS performance by saving the machine from having to do disk I/O and from having to work out how to perform a particular SQL statement more than once.

The golden rule is that the SGA must not be paged or swapped out. The amount of memory that can be allocated to the SGA depends on:

- The number of users. High numbers of users need more SGA but also require memory for their processes.
- The actual memory available.
- If the machine is a DB server or stand-alone.

Set the Oracle init.ora parameters:

- db_block_buffers
- shared_pool_size

For example, as a rough guide for initial sizing, see Table 17.

Table 17. SGA memory sizes

System type	Stand-alone percent of memory	Server only percent of memory
OLTP	30 percent	40 percent to 60 percent
DSS	40 percent to 70 percent	50 percent to 80 percent

The performance benefit is high, and there is no risk.

13.11.7 Oracle database writers

In the AIX hint for asynchronous I/O, it is recommended that this is switched on, but then you should set the Oracle parameters to:

For Oracle 7: `db_writers=1`

For Oracle 8: `dbwr_io_slaves=1`

Please refer to 13.6.4, “db_writers, db_writer_processes and dbwr_io_slaves” on page 307 for more information.

The performance benefit is medium and no risk.

13.11.8 Oracle buffer cache hit ratio tuning

To tune the Oracle SGA buffer cache you need to know the buffer cache hit ratio.

This can be determined by using the `utlbstat.sql` and `utlestat.sql` scripts. See 10.1.2, “Oracle monitoring tools” on page 215 for more information. Within the report.txt file, you will find all the statistics needed.

Then, calculate the hit ratio as:

```
1 - (physicalreads/(db block gets + consistent gets)) *100
```

As an alternative to the `utlbstat` and `utlestat` scripts to just get the statistics needed to do this calculation, use the SQL statement found in B.2.2, “Buffer cache hit ratio - manual” on page 374.

If you want to get Oracle to do the mathematics for you, then run the SQL statement found in B.2.3, “Buffer cache hit ratio - automatic” on page 374.

To change the size of the buffer cache, change the db_block_buffers Oracle parameter and restart Oracle.

If the cache hit ratio is low (below 80 percent), then increasing the buffers should increase performance.

The report.txt file also includes estimates on the effect of changes to the buffer cache.

Some DSS databases that read vast quantities of data into the cache may never have high cache hit ratios. Using the sort_direct_writes, sort_write_buffers, and sort_write_buffer_size parameters can help increase the hit ratio.

The performance benefit is high, and there is no risk.

13.11.9 Split the database disks from the AIX disks

The disk access patterns of a database, and that of the various parts of the AIX operating system, are very different. If they share disks, the database performance will slow down for unexpected reasons, such as users copying files or some paging taking place. If a disk contains both AIX and the database files, it also makes it much harder to identify the cause of a problem, and more advanced tools will be needed.

The performance benefit is high, and there is no risk.

13.11.10 Oracle redo log should have a dedicated disk

For databases that update and insert a large number of records, the redo log can rapidly become the bottleneck. The redo log is unique in the database because it is only a serial write I/O activity (the rest of the database tends to do random read and write). The log benefits from having a dedicated disk so that the disk heads are always at the right place.

The performance benefit is high, and there is no risk.

13.11.11 Mirror the redo log or use RAID 5 fast-write cache option

The redo log is vital for all recovery of the database in the case of a disk failure but is, in itself, a single point of failure, unless it has some sort of disk protection. It is write only. This is not suitable for a RAID 5 without the fast-write cache option. The other alternative is a disk mirror.

The performance benefit is high and no risk.

13.11.12 Oracle redo log groups or AIX mirrors

There are two methods of protecting the redo log for recovery purposes.

The first, historically, is using Oracle redo log groups. Oracle saves the log data to two or three different logs. Each log is identical, but this works on any Oracle supported platform. If one redo log disk fails, the other is available for recovery. See the Oracle Concepts manual for the full details.

The other way is to just use one redo log group (so Oracle only outputs one log), but protect this with AIX mirroring for recovery purposes.

Some sites use both, for example, three redo log groups each of which are mirrored. This results in six copies of the log, which seems like overkill and paranoia. This can be the results from physical layout policies that do not make sense for advanced operating systems, such as AIX. The argument is it protects from corruption from both Oracle and AIX, but, in most cases, the corruption would be found in all six copies.

We recommend only using AIX mirroring to protect the redo log. Using the Oracle redo log groups results in Oracle making multiple write system calls (one for each redo log group), which is less efficient.

The performance benefit is low, both methods work fine, and there are no risks.

13.11.13 Oracle parallel recovery

In the Oracle init.ora configuration file, set:

```
recovery_parallelism=[number of processors but not less than 2]
```

This means that the recovery, as the result of a system or Oracle failure, will be faster, as it will make use of all the processors on an SMP machine.

The performance benefit during recovery is high and no risk.

13.11.14 Oracle db_file_multiblock_read_count parameter

This feature is used by Oracle to read-ahead blocks when it is doing a sequential read through a file, such as full table scans. For example, if the file has one table in it or is within a range of blocks on the disk (the table was loaded that way), then a full table scan does sequential reads. AIX will also attempt read-ahead for JFS based files but not for raw device.

For OLTP type workloads that do random read and write disk I/O, this parameter will have no effect and is best left at the default value.

In the Oracle init.ora configuration file, set:

```
db_file_multiblock_read_count = [8, 16 or 32]
```

This should be set so that db_block_size * db_file_multiblock_read_count is greater than the LVM stripe size.

The performance benefit is medium, and there is no risk.

13.11.15 Oracle redo buffer latch

Set the following init.ora file parameters:

log_small_entry_max_size= 0 (removed in later versions of Oracle)

```
log_simultaneous_copies=[3 times the number of processors]
```

The performance benefit is medium, and there is no risk.

13.11.16 Oracle redo buffer size

Use the `utlbstat` and `utlestat` scripts to determine if the redo buffer has filled up and transactions have to wait for free space. See 10.1.2.4, “The UTLBSTAT/UTLESTAT monitoring tool” on page 217.

Look for the line with *redo log space requests*. If it is not there, then the value is zero, and the size is large enough. If this is more then 10, then it is important to increase the buffer size.

Alternatively, run the SQL statement in B.2.5, “Redo log buffer too small” on page 375.

The performance benefit is low unless the buffer is too small and no risk.

13.11.17 Oracle shared pool size

Run the SQL statement in B.2.4, “Shared pool free memory” on page 374 and check the amount of shared pool free memory in the SGA. Often the shared_pool is too large, as DBAs like to play safe.

If it is zero, then the dictionary cache and library cache hit ratios need to be determined. The library cache and dictionary cache are part of the shared pool, and the statistics are reported in the `utlbstat` and `utlestat` scripts (see 10.1.2, “Oracle monitoring tools” on page 215).

Understanding these ratios is complex, and the best reference is the *Oracle 8 Server Tuning* manual. If in doubt, make the shared pool size larger.

The performance benefit is medium and medium risk.

13.11.18 Oracle tablespace and table creation

Always create the tablespaces and tables with the optional storage options and, in particular, the INITIAL and NEXT options.

Always set the PCTINCREASE option to zero to stop new extents from being created with ever bigger sizes. Otherwise, this will eventually both waste disk space and make the next extent size so large it will not fit in the tablespace and cause a transaction insert to fail or data loads to fail unexpectedly.

Old rumors about keeping the number of extents to one or a low number for performance reasons are no longer true (within reason). Anything below 100 extents for large files (512 MB or more) is fine.

The performance benefit is medium and no risk.

13.11.19 Number of Oracle rollback segments

For OLTP systems, each transaction has to have an allocated rollback segment space to hold older copies of rows for database consistency during

the transaction and to roll back the transaction if it aborts. Investigate the roll back with the SQL statement in B.2.6, “Rollback segment” on page 375.

This details the active transactions using the roll back (XACTS) and the number of waits (WAITS) that transactions have had to endure.

Zero WAITS is the goal. This would mean no transaction have been blocked due to this resource. WAITS higher than zero indicates there are not enough rollback segments.The active transactions help you to decide if they are overused. Just one user is great, up to four is okay, but over that, it is likely to cause more problems.

Deciding the number of rollback segments to create is difficult. See 13.6.11, “rollback_segments” on page 309 for a little more information.

The performance benefit is medium and no risk.

13.11.20 Oracle parallelization

High parallelization and high numbers of users can cause a major bottleneck in the system, as too many processes are all requiring CPU, memory, and Disk I/O. If parallel queries are being used, then the amount of parallelization has to be predetermined and decided at the table or SQL level. However, when running, the effect can be monitored. If:

- Too low

  This results in the CPU and disks not reaching high utilization, and the response times are longer than necessary.

  When users are making peak use of parallel queries, check the CPU usage to determine if there is spare capacity and increase parallel_max_servers to suit.

- About right

  Look for nicely balanced system utilization - All the CPUs are 70 percent busy, no or little paging occurs, and the disks are less than 40 percent busy.

- Too high

  This results in CPU 100 percent utilized or disks above 60 percent utilized, poor response times for other work, and running out of parallel servers.

  The numbers of parallel servers is controlled with the parallel_min_servers and parallel_max_servers init.ora parameters.

  Check the use of parallel query servers with the following SQL:

```
select *
from v$pq_sysstat;
```

If the *Servers Busy* value is the same as parallel_max_servers, then this needs tuning. In this case, the amount of parallelization needs to be reduced.

The performance benefit is medium, and the risks are low.

13.11.21 Oracle archiver buffers

The log_archive_buffer_size effects the performance of the archiver, which is used to copy log files to other resources; so, redo log space can be reused later. Set the Oracle init.ora file parameters:

```
log_archive_buffer_size=[upto 128]
log_archive_buffer=[default of 4]
```

If the archive process uses large, and many, buffers, then its speed can be increased, but it can then take large amounts of CPU, memory, and disk I/O. This may effect the online user response time; so, be careful and monitor the effect of the archiver on performance of the system as a whole.

This may give 20 percent better performance of the archiver.

The performance benefit is medium, and the risks are low.

13.11.22 Oracle use TRUNCATE rather than DELETE all rows

If you need to remove all the rows from a table, it can take a long time to do this as part of a transaction with the `DELETE` statement. Oracle has the `TRUNCATE` statement that does this by removing all the extents from the table, which is an extremely efficient and fast way to remove the rows.

The performance benefit is large, and there are no risks.

13.11.23 Oracle marking and batch deleting rows

Many databases do not remove rows from the database, as this historical information can be useful, for example, for further sales and marketing analysis. Deleting rows can result in a database full of blocks that are not full of data any longer - a form of fragmentation.

One technique is not to remove rows at all but to have an extra column that signifies the row is no longer needed. Some databases use a deleted date and a column for who deleted it or why it should be deleted. Then, a batch job runs later to remove or achieve the data.

The performance benefit using this technique is small, and there are no risks.

13.11.24 Oracle SQL*Loader I/O buffers

While loading data with SQL*Loader, it ends up waiting for the disk I/O to complete. Increasing the SQL*Loader BUFFERS parameter will greatly improve load performance.

The performance benefit is high, and there are no risks.

13.12 Other tuning hints

This section includes a few further tuning hints for networking and compiling.

13.12.1 Network TCP/IP

SQL*Net V2 uses 2 KB packet sizes. The underlying packet size is 1 KB for most installations. The packet size can be changed with SQL*Net connection string parameters.

The performance benefit is medium, and the risks are high.

13.12.2 Compiling programs with embedded Oracle SQL

When compiling programs with embedded SQL statements (for example, Pro*C), use the best optimization level provided by the compiler. For AIX and the C compiler use: `-O`. This is the same as the `-O2` optimization level. Although `-O3` is available, it can make small code order changes and requires extra detailed application testing.

If you are compiling an application on one machine to run on that same machine or identical machines at the same AIX and Oracle version, we recommend you to use the default compiler options. If, however, you are compiling for many different RS/6000 machines (which may not be under your control), we recommend compiling for common mode. Use:

`-qarch=COM` for common mode (this will runs all RS/6000s)

Only use the following if you need that last 2 percent in performance and know that the code will never run on alternative hardware.

`-qarch=PWR` for POWER only machines

`-qarch=PWRX` for POWER2 only machines

`-qarch=PPC` for POWERPC only machines

The performance benefit is medium, and the risks are low.

13.13 Books for Oracle database administration and tuning

The aim of this section is to give you some comments on the Oracle books that are listed in the bibliography.

- *Oracle 8 Server Concepts* Volume 1 and 2 from Oracle Corporation, A54646, explains Oracle well.
- *Oracle 8 Server Tuning* from Oracle Corporation, A54638 is a good start in tuning Oracle but very theoretical and general.
- *Oracle Performance Tuning - Tips and Techniques*, ISBN 0-0788-2434-6, has excellent coverage, a lot of hands-on examples, and includes Oracle 8.
- *Oracle for AIX Performance Tuning Tips* from Oracle Corporation, A32146 is excellent for covering AIX specifics but was published in 1995.
- *Oracle 7 Performance Tuning Tips for UNIX* from Oracle Corporation, part number A22535 is a good summary of tuning Oracle plus the Oracle Flexible Architecture.
- *Oracle 8 & UNIX Performance Tuning* by Ahmed Alomari from Prentice Hall, ISBN 0-1390-7676-X. 310 pages. A good hands on tuning book with large OLTP and DSS tuning sections but does not cover AIX.
- *Oracle Performance Tuning* by Peter Corrigan and Mark Gurry from O'Reilly, ISBN 1-5659-2048-1, 600 pages. Good book covering application tuning too
- *Oracle Backup and Recovery Handbook* by Rama Velpuri from Osbourne/McGraw-Hill, ISBN 0-0788-2106-1. 380 pages. All you need to know about the subject.
- *OCP Training Guide: Oracle DBA* by Willard Baird II from New Riders, ISBN 1-5620-5891-6. 500 pages. Good all round coverage of DBA role including performance tuning.

Chapter 14. Austin - we have a problem!

In this chapter we detail what to do in case of a performance problem that you cannot solve or you think is not going to be solved by performance tuning of the RS/6000 hardware, the AIX operating system, the database or the application.

The structure is as follows

1. How to get and use *perfpmr* to collect performance data.
2. What you can do today in order to drastically reduce the time it will take to resolve a performance problem.
3. Information about how to raise a PMR in the most effective. way.
4. A list of common issues that cause problems.

The aim is to reduce the chances of a problem and to get the problem resolved sooner.

Note

This chapter assumes you have a support contract with IBM or an IBM Business Partner, so that you can open a Problem Management Record (PMR) to resolve the problem.

14.1 Perfpmr - the performance data collection tool

As most performance problems are complex, support will want a lot of information to allow them to check all the likely causes. To help in this AIX has a tool used to collect this data called *perfpmr*. Until AIX 4.3.2 this was supplied with your copy of AIX and can be found in the bos.perf.pmr fileset. Improvements are made to this tool regularly, based on experience of tracking down performance problems. From AIX 4.3.3 on perfpmr has to be obtained from the ftp site as described in the following section.

14.1.1 Get the latest version of perfpmr

To get the latest copy of perfpmr go to the following URL with your Internet browser:

`ftp://ftp.software.ibm.com/aix/tools/perftools/perfpmr/`

Then take the link to the version of AIX which is on the machine with the problem. Read the *readme* file and print it. Then download the *perfpmr*

package, it is roughly 150 KB in size. Copy these to the problem machine and follow the instructions in the *readme* file.

Some parts of the perfpmr data collection will only work if the bos.adt.samples and perfagent.tools filesets are loaded onto the machine. If these are missing (check with `lslpp -l bos.adt.samples perfagent.tools`) then less information will be collected. This fileset can be found on your AIX media and can be installed via `smitty installp`. We recommend loading this fileset.

If you are running the version from the ftp site, place the downloaded file in a sensible directory as the *root* user before running `perfpmr`. Uncompress the file with the `uncompress`, unpack the files with the `tar` command and add that directory to your PATH variable. Also, note this version is executed with the `perfpmr.sh` command and the output goes into the current directory.

A simple collect for 10 minutes (600 seconds) would need the following:

```
mkdir /tmp/perfdata
cd /tmp/prefdata
perfpmr.sh 600
```

Please make sure that there is enough space in the filesystem where you collect the performance data in (in our example it would be /tmp). Especially trace outputs can get huge, particularly on busy SMP systems. Allow a minimum of 25 MB up to 50 MB. On very large and busy systems you might need to allow more space.

Once finished it is worth looking in the *.sum files as these are the summary of the configuration and performance data. The config.sum file has the configuration details and can be useful in disaster recovery as it has all the filesystems, volume group and logical volume details. The monitor.sum file contains a summary of the performance data.

14.1.2 AIX media supplied version

If you are using the AIX media supplied version then do the following as the root user before running `perfpmr`:

```
export PATH=$PATH:/usr/sbin/perf/pmr
```

This version is executed via the `perfpmr` command and will place the results in /var/tmp/perf.

However, it is not recommended to use this version of perfpmr because it collects less data than the latest version from the ftp server. The shipment of the bos.perf.pmr fileset is discontinued with AIX 4.3.3 and further releases.

14.2 Before you have a problem

One of the hardest issues in resolving a performance problem is determining what has changed on the machine that has caused the problem. This can be made simple by comparing the configuration and performance data of the machine with and without the problem. This can only be achieved when you have captured the performance data before the problem arises.

The Austin Performance team therefore recommends that customers collect perfpmr data before and after making changes to the system. Any change to the following can effect performance:

- Hardware configuration - adding, removing or changing things such as how the disks are connected
- AIX - installing or updating a fileset, installing PTFs and changing parameters
- Database - installing new versions and fixes
- Database - configuration or changing data placement
- Database tuning
- Application changes
- Tuning options in AIX, RDBMS or application

One option is to run `perfpmr` before and after each change. The alternative is running `perfpmr` at regular intervals like once a month and save the output. When a problem is found, the previous capture can be used for comparison. It is worth collecting a series of perfpmr outputs in order to support the diagnostics of a possible performance problem.

The Austin Performance team also recommends collecting perfpmr data for various periods of the working day, week or month when performance is likely to be an issue. For example you may have workload peaks:

- in the middle of the mornings for online users
- during a late night batch run
- during the end of month processing
- during major data loads.

Collect perfpmr data for each of these peak in workload as a performance problem might only cause problems during one of these periods and not during other times.

14.3 Raising a Problem Management Record (PMR)

If you are going to raise a problem via your support channel then it is worth preparing, in advance, the information that you will be asked to supply to allow the problem to be investigated. Your local support people will attempt to solve your performance problem directly with you and quickly. But if it is a complex problem (and performance problems frequently are) then it will eventually be escalated to IBM Austin - the home of the RS/6000 and AIX development.

First, note that PMRs will be given a severity:

Severity 1 The production system is not available and someone, for instance the system administrator, has to be available for assistance 24 hours a day.

Severity 2 The system has reduced function and someone will be available during the day.

Severity 3 All other problems.

Severity 4 Requests for information.

So only request a Severity 1 problem if you are willing to have people available 24 hours a day to assist the gathering of information and implementing tools and a solution. Also make sure that you supply:

- the name of the technical person working on this problem who has root access to the machine involved
- the telephone number
- an e-mail address
- that you have FTP access to the Internet for downloading updates
- access to fixdist for downloading PTFs

Next get the latest version of `perfpmr` see Part 14.1, “Perfpmr - the performance data collection tool” on page 345 and collect the performance information. You can then supply this data when requested and even volunteer to provide this information when opening the PMR in order to save time. Often support will ask you to ftp the perfpmr output to them.

Three further ways you can help to get the problem resolved faster are:

1. Provide a clear written statement of the problem but be sure to separate the symptoms and facts from the theories, ideas and your own conclusions. PMRs that report - *"the system is slow"* are going to require a

lot further investigation to simply workout what you mean by *slow*, how it is measured, and what is acceptable performance.

You should try to provide all the information detailed in the Table 18 on page 349 at the start. This will save a great deal of time.

2. Confess immediately everything that has changed on the system in the weeks before the problem - missing out something that changed will block a possible directions for the team to investigate and will only delay finding a resolution. Do not hide facts because you think they should not effect performance until later. If all the facts are available, the performance team can eliminate the unimportant ones quickly.
3. Supply information on the correct machine. In very large sites it is easy to accidentally collect the data on the wrong machine - this makes it very hard to investigate the problem.

14.3.1 PMR information

This table is to help you gather all the information about the machine, configuration and performance to allow the experts to investigate the problem without spending time asking lots of questions. You might want to photocopy this table and filling in the answers.

Table 18. PMR basic information

Area	Details	Your answers
RS/6000 Basics	Model	
	RAM	
	Number and type of disks (include any documentation you have on the disk layout)	
Versions	AIX	
	Database	
	Application	
Workload type	OLTP, DSS, Batch, Web, other	
Network	Number and type of networks	

Area	Details	Your answers
Symptoms	System error log entries	
	Database error log entries	
	Error messages	
	Original response time and current response time	
	Is everything slow or just some things?	
	Does the slow behavior also occur when the system is idle?	
	Original throughput and current throughout	
	Do processes hang?	
	Does the system hang or did it crash? How did you recover from this situation?	
Can the problem be demonstrated with the execution of a specific command or sequence of events? If client/server, can the problem be demonstrated when run just locally on the server?		
System dump	Setup a system dump or even have one available	
Normal behavior	State what you normally expect	
Performance	CPU percent busy	
	SMP - are all CPUs used?	
	Disks average percent busy	
	Disks top disk percent busy	

Area	Details	Your answers
Actions	What you have tried?	
	Anything you changed?	
	Any ideas you have	
	Anything unusual you have seen	
	Does rebooting the system make the problem disappear for a while?	

14.4 Most common sources of database performance PMRs

Below are the causes of the most common issues raised as database performance PMRs to the IBM Austin team. We include this list as it might help you avoid one of these problems:

- Maximize the database's use of real memory but do not cause paging on the system. Also lowering the AIX filesystem buffer cache maximum size can help.
- Incremental upgrades and changes allow the quickest diagnosis.
  - Do not upgrade AIX, the database and the application all together.
  - Do not change software at the same time as changing hardware configurations such as the disk layout.
  - Do not change software at the same time as changing database parameters or data placement.
- AIX databases generally benefit from using raw devices, if this does not cause backup problems.
- SSA disks are not faster than SCSI disks. The SSA subsystem supports more disks, are easier to configure and manage, and the interface allows more thoughput, but the disks are the same.
- RAID 5 frequently has poor write performance but may be difficult to notice initially because the write cache on the device might help for a certain amount of updates. However, this problem is solved, to a large extent, if the latest SSA Advanced RAID adapters with the fast-write cache option are used.
- SMP systems allow higher capacity but single threaded transactions will run no faster than 1 processor allows. To make use of SMP systems,

single large tasks need to be broken down into parts that can be executed concurrently to make use of the whole machine.

14.5 Avoiding the next performance crisis

Once the problem is resolved, try to think of a way in which this situation can be avoided next time.

For example:

- a test system to check software combinations and tuning options
- a better regression test after loading new versions, features or bug fixes
- the means to quickly remove a change to the system
- extra disk space to allow two versions to co-exist
- better change control procedures
- collecting before and after performance data (see Part 14.2, “Before you have a problem” on page 347)
- further education and training to enhance skills

Appendix A. AIX performance tools summary

This appendix is a summary, meant as a quick reference, of the most important AIX commands that can help in monitoring and tuning RDBMS performance on AIX. Many of these commands, including the commands that update AIX parameters, require *root* permission to run. Only the most commonly used options are provided for each command. Please consult the following references for more detailed information related to monitoring AIX system performance:

- *AIX V 4.3 Commands Reference,* SBOF-1877
- *AIX Performance Tuning Guide,* SC23-2365
- *Understanding IBM RS/6000 Performance and Sizing* redbook, SG24-4810
- *RS/6000 Performance Tools in Focus* redbook, SG24-4989

AIX standard documentation can be found at:
`http://www.rs6000.ibm.com/doc_link/en_US/a_doc_lib/aixgen/`

IBM Redbooks can be found at:
`http://www.redbooks.ibm.com`

A.1 Summary of performance bottlenecks

The guidelines contained in Table 19 can be used to determine potential bottlenecks on your system and the referenced tools can be used to determine the current values of the thresholds. If the values are over one of the listed thresholds, it is possibly a feature of your configuration or application, so do not assume that a performance problem exists. The most reliable input on system response times most often comes from the users of the system.

Table 19. Performance bottleneck thresholds

CPU bound	when %user + %sys greater than 80% `(vmstat)`
Disk I/O bound	when %iowait greater than 40%[1] (AIX 4.3.3 or later) `(vmstat)`
Application disk bound	when %tm_act greater than 70% `(iostat)`
Paging space low	when paging space greater than 70% active `(lsps -a)`

Paging bound	paging logical volumes %tm_act greater than 30% of the I/O (iostat) and paging activity greater than 10 * the number of CPUs (vmstat)
Thrashing	rising page outs, CPU wait and run queue high (vmstat and sar)
[1] Advanced performance tools like filemon should be used in order to determine if the system is really I/O bound.	

A.2 filemon - File I/O Monitor

The filemon command is used to monitor the performance of the file system and report the I/O activity on behalf of files, virtual memory segments, logical volumes, and physical volumes. The global reports list the most active files, segments, logical volumes, and physical volumes during the measured interval. They are shown at the beginning of the filemon report. By default, the logical file and virtual memory reports are limited to the 20 most active files and segments, respectively, as measured by the total amount of data transferred. If the -v flag has been specified, activity for all files and segments is reported. All information in the reports is listed from top to bottom as most active to least active.

Syntax: `filemon -i file -o file -d -Tn -P -v _O levels`

Example:`filemon -O all -o file.out`
Start workload (in a production system workload is usually already present) and then stop trace activity with `trcstop`.

Most Active Files report

Column	Description
#MBS	Total number of megabytes transferred to/from file. The rows are sorted by this field, in decreasing order.
#opns	Number of times the file was opened during measurement period.
#rds	Number of read system calls made against the file.
#wrs	Number of write system calls made against the file.
file	Name of the file (the full path name is in the detailed report).
volume:inode	Name of the logical volume that contains the file, and the file's i-node number. This field can be used to associate a file with its corresponding persistent segment, shown in the virtual memory I/O reports. This field may be blank; for example, for temporary files that are created and deleted during execution.

Most Active Segments report

Column	Description
#MBS	Total number of megabytes transferred to/from segment. The rows are sorted by this field, in decreasing order.
#rpgs	Number of 4096-byte pages read into segment from disk (page in).
#wpgs	Number of 4096-byte pages written from segment to disk (page out).
segid	Internal ID of segment.
segtype	Type of segment: working segment, persistent segment (local file), client segment (remote file), page table segment, system segment, or special persistent segments containing file system data (log, root directory, .inode, .inodemap, .inodex, .inodexmap, .indirect, .diskmap).
volume:inode	For persistent segments, name of logical volume that contains the associated file, and the file's inode number. This field can be used to associate a persistent segment with its corresponding file, shown in the file I/O reports. This field is blank for non-persistent segments.

Note

The virtual memory analysis tool, svmon can be used to display more information about a segment, given its segment ID (segid), as follows: svmon -S <segid>.

Most Active Logical Volumes report

Column	Description
util	Utilization of the volume (fraction of time busy). The rows are sorted by this field, in decreasing order.
#rblk	Number of 512-byte blocks read from the volume.
#wblk	Number of 512-byte blocks written to the volume.
KB/sec	Total transfer throughput, in Kilobytes per second.
volume	Name of volume.
description	Contents of volume: either a file system name, or logical volume type (paging, jfslog, boot, or sysdump). Also, indicates if the file system is fragmented or compressed.

Most Active Physical Volumes report

Column	Description

Most Active Physical Volumes report

`util`	Utilization of the volume (fraction of time busy). The rows are sorted by this field, in decreasing order.
`#rblk`	Number of 512-byte blocks read from the volume.
`#wblk`	Number of 512-byte blocks written to the volume.
`KB/sec`	Total volume throughput, in Kilobytes per second.
`volume`	Name of volume.
`description`	Type of volume, for example, 120MB disk, 355MB SCSI, or CDROM SCSI

A.3 iostat - Disk I/O Statistics

The `iostat` command is used to report CPU and I/O statistics for TTY devices, disks, and CD-ROMs. It is used to generate reports that can be used to change the system configuration to better balance the input/output load between physical disks.

Syntax: `iostat interval count`

Flag	Meaning
`interval`	number of second between outputs
`count`	number of times to output
Examples:	
`iostat 10 20`	20 lines output with 10 seconds between each
Report Output:	
%tm_act	Percentage of time active
Kbps	Kilobytes per second transferred
tps	Transfers per second
msps	Milliseconds per seek (if available)
Kb_read	Total Kilobytes read (likewise for write)

A.4 lsattr - List attributes

The `lsattr` command lists the attributes of AIX resources. The `sys0` resource includes performance statistics.

Syntax: `lsattr -El sys0`

Examples:

`lsattr -El sys0`	outputs details of AIX parameters including minpout and maxpout

A.5 lscfg - List configuration

The `lscfg` command lists the details of the machine.

Syntax: `lscfg [-v]`

Flag	Meaning
`-v`	outputs the full details

Examples:

`lscfg`	outputs details of the machine
`lscfg -v`	outputs the full details, part numbers and levels

A.6 lsdev - List devices

The `lsdev` command lists the details of the devices in the machine.

Syntax: `lsdev -C`

Syntax: `lsdev -Cc class`

Flag	Meaning
`-Cc`	outputs device details for one class only

Examples:

`lsdev -C`	outputs details of all devices
`lsdev -Cc class`	output of a particular class (memory, disk, tape, ...)

A.7 lslpp - List licensed program produce

The `lslpp` command lists the packages, filesets and files loaded in the AIX system.

Syntax: `lslpp [-lLa <fileset>] [-f <fileset>] [-w <filename>]`

Flag	Meaning

`-l <fileset>`	outputs the most recent levels of the fileset
`-La <fileset>`	outputs the full details and updates of the fileset
`-f <fileset>`	outputs the files within a fileset
`-w <file>`	outputs the fileset the file belongs too
Examples:	
`lslpp -l "bos.rte.*"`	outputs levels of this fileset
`lslpp -La "bos.rte*"`	outputs above plus update information
`lslpp -f  "bos.rte"`	outputs the files of this fileset
`lslpp -w "/usr/bin/vi"`	outputs the fileset this file belongs too
`lslpp -w "*installp*"`	outputs the files that contain any filename that includes the directory or f ilename installp

A.8 lslv - List logical volume

The `lslv` command lists the details of the logical volume and their placement on the disks.

Syntax: `lslv [-l] <volume group name>`

Flag	Meaning
`-l`	outputs the placement on the disks
Examples:	
lslv lv00	outputs details of the logical volume
lslv -l lv00	outputs the placement of the logical volume on disks (physical volumes)

A.9 lsps - List Paging Space

The `lsps` command displays the characteristics of paging spaces, such as the paging space name, physical volume name, volume group name, size, percentage of the paging space used and whether the space is active or inactive.

Syntax: `lsps -a -s [paging space]`

Flag	Meaning
`-a`	displays all paging spaces

`-s`	displays summary of all paging spaces
Examples:	
`lsps -a`	lists the characteristics of all paging spaces

A.10 lspv - List physical volume

The `lspv` command lists the details and contents of physical volumes (disks).

Syntax: `lsvg [-p] [-l] <hdisk name>`

Flag	Meaning
`-p`	outputs contents and placement
`-l`	outputs contents of the physical volume
Examples:	
`lspv hdisk0`	outputs volume group names only
`lsvg -l hdisk22`	outputs details about disk hdisk22
`lsvg -p hdisk22`	outputs more details about disk hdisk22

A.11 lsvg - List volume group

The `lsvg` command lists the names of the volume group, their contents and their details.

Syntax: `lsvg [-i] [-l] <volume group name>`

Flag	Meaning
`-i`	takes volume group names from standard input
`-l`	outputs the details of a volume group
Examples:	
lsvg	outputs volume group names only
lsvg rootvg	outputs the details of the volume group called rootvg
lsvg -l rootvg	outputs details of volume group called rootvg

A.12 ncheck - Inode Check

The `ncheck` command is used to display the i-node numbers and path names for filesystem files.

Syntax: `ncheck [-a][-i inodenumber...] [-s] [filesystem]`

Flag	Meaning
`-a`	lists all filesystems including those starting with '.' and '..'
`-i inode`	finds the file(s) with these inode numbers
`-s`	lists special and set UID files
Examples:	
`ncheck -a /`	lists all files in the '/' filesystem
`ncheck  -i 2194 /tmp`	finds the name for inode 2194 in /tmp

A.13 netpmon - Network Monitor

The `netpmon` command is used to monitor and report on network activities and network related CPU usage. It uses the AIX *system trace* to gather information.

Syntax: `netpmon -o file -Tn -P -v -O report-type`

Flag	Meaning
`-o outputfile`	puts the output to a file, not stdout
`-T n`	sets the output buffer size (default 64000)
`-P`	forces the monitor process into pinned memory
`-v`	Verbose (default only top 20 processes)
`-O`	Allows the selection of one of the following options: cpu, dd(device driver), so(socket), nfs, all

Example: `netpmon -O all -o net.out`
Start workload (see filemon) and then stop trace activity with `trcstop`.

A.14 nfsstat - Network File System statistics

The `nfsstat` command lists the NFS details.

Syntax: `nfsstat`

Examples:	
`nfsstat`	outputs all NFS statistics

A.15 nmon - online monitor

The `nmon` command is used to display all the AIX statistics on one screen and updates them every 2 seconds. When running hit `h` for further help on the options or hit `q` to quit. An alternative mode saves the same data to a file that can be loaded into a spreadsheet.

Syntax: `nmon [-?][-fdt]`

Flag	Meaning
`-?`	output help information on running `nmon`
`-fdt`	run in file output mode, including disk and top process statisitics

Note: this tool is not supported by IBM and no warranty is given or implied by including this tool in this redbook. It is available to IBM at:

`http://w3.aixncc.uk.ibm.com`

and to IBM Business Partners via PartnerInfo.

A.16 no - Network options

The `no` command lists the details of the network options.

Syntax: `no -a`

Flag	Meaning
`-a`	outputs all options
Examples:	
`no -a`	outputs all network options

A.17 ps - Process State

The `ps` command is used to display the status of currently active processes.

Syntax: `ps -a -e -f -l -p plist -u user -x .`

Flag	Meaning
`-a`	writes information about all processes, except the session leaders and processes not associated with a terminal to standard output

`-e`	lists every user's process
`-f`	full listing
`-l`	long listing
`-p pid`	lists the process number N
`-u user`	lists the specified user's processes (-u fred)
Examples:	
`ps -fu jim`	Lists user jim's processes in full
`ps -lt 04`	List all processes on terminal tty04
`ps -fe`	List all processes

Report column headings	**Meaning**
PID/PPID	Process IDentity & Parent Process IDentity
S	State= Running, Sleeping, Waiting, Zombie, Terminating, Kernel, Intermediate
UID/USER	User IDentity/User name
C	CPU recent use value (part of priority)
STIME	Start time of process
PRI	Priority (higher means less priority)
NI	NIce value (part of priority) default 20
ADDR	ADDRess, of stack (segment number)
SZ	SiZe of process in 1K pages
CMD	Command the user typed (`-f` to display more)
WCHAN	Event awaited for (kernel address)
TTY	Terminal processes connected to (- = none)
TIME	Minutes and Seconds of CPU time consumed by the process
SSIZ	Size of kernel stack
PGIN	# of pages paged in
SIZE	Virtual size of data section in 1K's
RSS	Real memory (resident set) size of process 1K's
LIM	Soft limit on memory xx=none
TSIZ	Size of text (shared text program) image
TRS	Size of resident set (real memory)

%CPU	Percentage of CPU used since started
%MEM	Percentage of real memory used

A.18 rmss - Reduced Memory System Simulator

The `rmss` command is used to simulate a system with various sizes of real memory that are smaller than the actual amount of physical memory installed on the machine.

Syntax: `rmss -p -c M -r`

Flag	Meaning
`-p`	print the current value
`-c M`	changes to size M (in Mbytes)
`-r`	restores all memory to use
Examples:	
`rmss -c 32`	Change available memory to 32 Mbytes
`rmss -r`	Undo the above

A.19 sar - System Activity Reporter

The `sar` command is a standard UNIX command, used to gather statistical data about the system.

Syntax: `sar -A -o savefile -f savefile -i secs -s HH[:MM[:SS]] -e HH[:MM[:SS]] -P ALL interval number`

Flag	Meaning
-A	All stats to be collected/reported
-o savefile	Collect stats to binary file
-f savefile	Report stats from binary file
-i secs	Report at secs interval from binary file
-s and -e	Report stats only between these times
-P ALL	Report on all CPU stats
CPU related output	
%usr %sys	Percent of time in user / kernel mode

%wio %idle	Percent of time waiting for disk io/idle
Buffer Cache related output	
bread/s bwrit/s	Block I/O per second
lread/s lwrit/s	Logical I/O per second
pread/s pwrit/s	Raw disk I/O (not buffer cached)
%rcache %wcache	Percentage hit on cache
Kernel related Output	
exec/s fork/s sread/s swrite/s rchar/s wchars/s scall/s	Calls of these system calls per second exec and fork are used for process creation sread/swrite system calls (files, raw, tty or network). rchar/wchar the numbers of characters transferred *scall* is the total number of system calls per second
msg/s sema/s	Inter-process communication (IPC) for messages and semaphores
kexit/s ksched/s kproc-ov/s	Process exits, process switches and process overload (hit proc thresholds)
runq-sz	Average process on run queue
%runocc	Percentage of time with process on queue
swap-sz	Avgerage process waiting for page in
%swap-occ	Percentage of time with process on queue
cycles/s	# of page replace search of all pages
faults/s	# of page faults
slots	# of free pages on paging spaces
odio/s	# of non-paging disk I/O per second
file-ov, proc-ov	# of times these tables overflow per second
file-sz inode-sz proc-sz	Entries in the tables
pswch/s	Process switches per second
canch/s outch/s rawch/s	Characters per second on terminal lines
rcvin/s xmtin/s	Receive and transmit interrupts per second

Examples:

sar 10 100	Reports now at 10 second intervals

sar -A -o fred 10 6 >/dev/null	Collects data into fred
sar -A -f fred	Reports on the data
sar -A -f fred -s 10:30 -e 10:45	Reports for 15 minutes starting at 10:30 a.m.
sar -A -f fred -i60	Reports on a 1 minute interval rather that 10 seconds as collected
sar -P ALL 1 10	Reports on each CPU or the next 10 seconds

A.20 schedtune - Process Scheduling Tuning

The `schedtune` command is used to set the parameters for the CPU scheduler and Virtual Memory Manager processing.

Syntax: `schedtune -h sys -p proc -w wait -m multi -e grace -f ticks -t time_slice -D` (default)

Flag	Meaning
`-h 6`	Sets system wide criteria for when process suspension begins and ends (thrashing)
`-p 4`	Sets per-process criteria for determining process suspension begins and end
`-w 1`	Seconds to wait before trashing ended
`-e 2`	Seconds exempt after suspension
`-f 10`	Clock tick waited after fork failure
`-t 0`	Clock tick interrupts before dispatcher called
`-D`	Restore default values

Examples:

`schedtune -t5`	set time slice to 50 ticks
`schedtune`	Report current settings

A.21 svmon - System Virtual Memory Monitor

The `svmon` command is used to capture and analyze a snapshot of virtual memory.

Syntax: `svmon -G -Pnsa pid... -Pnsa[upg][count] -S sid... -i secs count`

Flag	Meaning

`-G`	Global report
`-P[nsa] pid..`	Process report n=non-sys s-system a=both
`-S[nsa][upg][x]`	Segment report nsa as above plus u = real memory, p = pinned, g = paging, x = top x items
`-S sid...`	Segment report on particular segments
`-i secs  count`	Repeats report at interval second & count times
`-D sid...`	Detailed report

`Detailed Report Output:`

Report column headings	**Description**
`size`	in pages (4096)
`inuse`	in-use
`free`	not inuse included rmss pages
`pin`	pined (locked by application)
`work`	pages in working segments
`pers`	pages in persistent segments
`clnt`	pages in client segments
`pg space`	paging space

Note: pages can be in more than one process

`Examples:`

`svmon -G`	Global / General statistics
`svmon -Pa 215`	Processes report for process 215
`svmon -Ssu 10`	Top ten system segments in real-mem order
`svmon -D 340d`	Detailed report on a particular segment

A.22 vmstat - Virtual Memory Management Statistics

The `vmstat` command is used to report statistics about kernel threads in the run and wait queues, memory, paging, disks, interrupts, system calls, context switches, and CPU activity. If the vmstat command is used without any options or only with the interval and optionally, the count parameter, like `vmstat 2`, then the first line of numbers is an average since system reboot.

Syntax: `vmstat interval count`

Flag	Meaning
interval	number of seconds between outputs
count	number of times to output
Report column headings	**Description**
`r`	# of processes on run queue per second
`b`	# of processes awaiting paging in per second
`avm`	active virtual memory pages in paging space
`fre`	real memory pages on the free list
`re`	Page reclaims, free but claimed before reused
`pi`	paged in (per second)
`po`	paged out (per second)
`fr`	pages freed (page replacement per second)
`sr`	pages per second scanned for replacement
`cy`	complete scans of page table
`in`	device interrupts per second
`sy`	system calls per second
`cs`	CPU context switches per second
`us`	User CPU time percentage
`sys`	System CPU time percentage
`id`	CPU idle percentage (nothing to do)
`wa`	CPU waiting for pending local Disk I/O
Examples:	
`vmstat 10 20`	20 lines output with 10 seconds between each

Special Considerations about vmstat on AIX V4.3.2 and earlier versions and AIX V4.3.3

AIX 4.3.3 contains an enhancement to the method used to compute the percentage of CPU time spent waiting on disk I/O (*wio* time). The method used in AIX 4.3.2 and earlier versions of AIX can give an inflated view of *wio* time on SMPs in some circumstances. The *wio* time is reported by the commands sar (%wio), vmstat (wa) and iostat (%iowait).

Method used in AIX 4.3.2 and earlier AIX versions
At each clock interrupt on each processor (100 times a second in AIX), a determination is made as to which of four categories (usr/sys/wio/idle) to place the last 10 ms of time. If the CPU was busy in usr mode at the time of the clock interrupt, then *usr* gets the clock tick added into its category. If the CPU was busy in kernel mode at the time of the clock interrupt, then the *sys* category gets the tick. If the CPU was NOT busy, then a check is made to see if ANY I/O to disk is in progress. If any disk I/O is in progress, then the *wio* category is incremented. If NO disk I/O is in progress and the CPU is not busy, then the *idl* category gets the tick. The inflated view of *wio* time results from all idle CPUs being categorized as *wio* regardless of the number of threads waiting on I/O. For example, RS/6000 with just one thread doing I/O could report over 90 percent *wio* time regardless of the number of CPUs it has.

Method used in AIX 4.3.3
The change in AIX 4.3.3 is to only mark an idle CPU as *wio* if an outstanding I/O was started on that CPU. This method can report much lower *wio* times when just a few threads are doing I/O and the system is otherwise idle. For example, an RS/6000 with four CPUs and one thread doing I/O will report a maximum of 25 percent *wio* time. An RS/6000 with 12 CPUs and one thread doing I/O will report a maximum of 8.3 percent 'wio' time.

A.23 vmtune - Virtual Memory Tuning

The `vmtune` command is used to modify the AIX Virtual Memory Manager (VMM) parameters for the purpose of changing the behavior of the memory management subsystem.

Syntax: `vmtune -p min -P max -f min -F max -r min -R max`

Flag	Meaning
`-p min`	min percentage of memory reserved for file pages (default 20 percent)
`-P max`	max percentage of memory reserved for file pages (default 80 percent)
`-f min`	number of pages on free list, below which page stealing starts (default 120)
`-F max`	number of pages on free list, above which page stealing stops (default 128)
`-r min`	min number of pages to be read ahead after sequential access is detected
`-R max`	max number of pages to be read ahead after sequential access is detected

Appendix B. Vital SQL

This appendix contains a list of vital Oracle and DB2 UDB SQL commands commonly used on a day-by-day monitoring task. These commands do not represent all the possible combinations, but they suggest the easiest way to collect the basic information on an RDBMS.

B.1 DB2 UDB

DB2 UDB provides several commands that query the catalog tables and the control files in order to retrieve a fast and valuable information output for the database administrator.

B.1.1 List the existing tables on a database

```
list tables for all
```

or

```
Select substr(tabschema,1,8) as "Qualified Name",
substr(tabname,1,50) as "Table name",
CASE type
             WHEN 'A' THEN 'Alias'
             WHEN 'H' THEN 'Hierarchy Table'
             WHEN 'N' THEN 'Nickname'
             WHEN 'S' THEN 'Summary Table'
             WHEN 'T' THEN 'Table'
             WHEN 'U' THEN 'Typed Table'
             WHEN 'V' THEN 'View'
             WHEN 'W' THEN 'Typed View'
END as "Table Type",
CASE status
             WHEN 'N' THEN 'Normal'
             WHEN 'C' THEN 'Check Pending'
             WHEN 'X' THEN 'Inoperative'
END as "Table Status"
from syscat.tables
```

B.1.2 Describe the structure of the columns in a table

```
describe table schema.table_name show detail
```

B.1.3 Describe the indexes defined in a table and their structure

```
describe indexes for table schema.table_name show detail
```

B.1.4 Describe structure of the columns within a SELECT statement

```
describe select column1, column2,..., columnX from schema.table_name
```

B.1.5 List all the tablespaces of a database

```
list tablespaces
```

or

```
Select tbspace as "Table Space Name",
tbspaceid as "Identifier",
CASE tbspacetype
             WHEN 'S' THEN 'System Managed Space'
             WHEN 'D' THEN 'Database Managed Space'
             END as "Table Space Type",
CASE datatype
WHEN 'A' THEN 'Permanent Data'
WHEN 'L' THEN 'Long Data'
WHEN 'T' THEN 'Temporary Table'
END as "Type of Data Stored"
 from syscat.tablespaces
```

B.1.6 List tablespace name, Id number, size, and space consumption

```
list tablespaces show detail
```

B.1.7 List the tablespace containers

```
list tablespace containers for tablespace_id show detail
```

B.1.8 Enable all monitor switches

```
UPDATE MONITOR SWITCHES USING bufferpool on;
UPDATE MONITOR SWITCHES USING lock on;
UPDATE MONITOR SWITCHES USING sort on;
UPDATE MONITOR SWITCHES USING statement on;
UPDATE MONITOR SWITCHES USING table on;
UPDATE MONITOR SWITCHES USING uow on;
```

B.1.9 Disable all monitor switches

```
UPDATE MONITOR SWITCHES USING bufferpool off;
UPDATE MONITOR SWITCHES USING lock off;
UPDATE MONITOR SWITCHES USING sort off;
UPDATE MONITOR SWITCHES USING statement off;
UPDATE MONITOR SWITCHES USING table off;
UPDATE MONITOR SWITCHES USING uow off;
```

B.1.10 Check the monitor status

```
get monitor switches
```

B.1.11 Reset the monitor counters for a specific database

```
reset monitor for database database_name
```

B.1.12 Show the locks existing on a database

```
get snapshot for locks on database_name
```

B.1.13 List application number, status, idle time, and AIX processes

```
db2 get snapshot for application on database_name | grep -E
"handle|thread|idle|status"
```

or

```
list applications show detail
```

Lists the aplication number, the current status, the idle time, and the associated AIX processes.

B.1.14 List connected and effectively executing users

```
db2 get snapshot for database manager | grep connections
```

Lists the users that are connected to a database and that are effectively executing.

B.1.15 Display the amount of memory being used for sort operations

```
db2 get snapshot for database manager | grep Sort
```

B.1.16 Display the number of deadlocks and lock escalations

```
db2 get snapshot for applications on database_name | grep -E "Application
Handle|Deadlock|escalation"
```

Displays the number of deadlock and lock escalations for each application.

B.1.17 Display the number of attempted SQL COMMIT statements

```
db2 get snapshot for all on database_name | grep "Commit statements
attempted"
```

B.2 Oracle

Oracle SQL commands can be issued directly using Server Manager or SQLPlus. An other way is to use predefined input files, that generate reports which make the output easier to read and understand, and thus facilitate the monitoring process.

B.2.1 Oracle number of transactions

```
SELECT value,name
FROM v$sysstat
WHERE statistic# <7 ;
```

This query gives you the number of transaction commits/aborts and other useful data from this well hidden internal Oracle table.

B.2.2 Buffer cache hit ratio - manual

```
select name, value
from v$sysstat
where name in ('db block gets', 'consistent gets', 'physical reads');
```

Then calculate: hit ratio as:

```
1 - ( physicalreads/(db block gets + consistent gets)) *100
```

B.2.3 Buffer cache hit ratio - automatic

```
select (1 - ( sum(decode(name, 'physical reads', value, 0))/
             ( sum(decode(name, 'db block gets', value, 0)) +
               sum(decode(name, 'consistent gets', value, 0)))))
          * 100 "Hit Ratio"
from v$sysstat;
```

This outputs the Hit Ratio directly but please use a script to run this SQL statement as it is complex and hard to type in correctly.

B.2.4 Shared pool free memory

```
select *
from v$sgastat
where name = 'free memory';
```

This outputs used memory from the shared pool, so compare it to the `shared_pool_size` init.ora parameters.

B.2.5 Redo log buffer too small

```
select name, value
from v$sysstat
where name = 'redo log space requests';
```

This outputs the number of times the redo log buffer was found to be full and the transaction waited for free space.

B.2.6 Rollback segment

```
select name, extents, rssize, xacts, waits, gets, optsize, status, status
from v$rollname a, v$rollstat b
where a.usn = b.usn;
```

This outputs the number of extents, their size, and the usage of the rollback segments. If `WAITS` is more than one then more rollback segments are needed. `XACTS` is the number of transactions actually using the rollback.

B.2.7 Oracle nested explain plan

```
select lpad(' ',2*level)||operation||' '||options||' '||object_name
query_plan
from plan_table where statement_id = 'xx'
connect by prior id = parent_id and statement_id = 'xx'
start with id =1;
```

B.2.8 Oracle report on tablespaces

```
spool tablespace.lst
set pagesize 999;

ttitle center 'TableSpaces' skip 1 -
       center '===========' skip 2;
column tablespace_name heading 'TableSpace' format A12;
column initial_extent heading 'Initial|Extent' format 999,999,999;
column next_extent heading 'Next|Extent' format 999,999,999;
column min_extents heading 'Min|Extent' format 999;
column max_extents heading 'Max|Extent' format 999;
column pct_increase heading '%|Increase' format 999;
column status heading 'Status' format A8;

select
 TABLESPACE_NAME,
 INITIAL_EXTENT,
 NEXT_EXTENT,
 MIN_EXTENTS,
 MAX_EXTENTS,
```

```
 PCT_INCREASE,
 STATUS
from user_tablespaces;
ttitle off;

ttitle center 'TableSpaces Sizes' skip 1 -
       center '================' skip 2;
select TABLESPACE_NAME,sum(BYTES) from dba_data_files group by
TABLESPACE_NAME;
```

B.2.9 Oracle report on tables

```
spool table.lst
set pagesize 999;

ttitle center 'Table Sizes' skip 1 -
       center '===========' skip 2;
column table_name heading 'Table' format A18;
column tablespace_name heading 'TableSpace' format A10;
column initial_extent heading 'Initial|Extent' format 999,999,999;
column next_extent heading 'Next|Extent' format 999,999,999;
column NUM_ROWS heading 'Num_rows' format 9,999,999,999;
column AVG_ROW_LEN heading 'Avg-Len' format 99,999;
column BLOCKS heading 'Blocks' format 99,999,999;
column EMPTY_BLOCKS heading 'Empties' format 999,999;
column DEGREE heading 'Degree' format A10;
column INSTANCES heading 'Instances' format A10;
column min_extents heading 'Min|Extent' format 999;
column max_extents heading 'Max|Extent' format 999;
column pct_increase heading '%|Increase' format 999;
column status heading 'Status' format A8;

select
 TABLE_NAME,
 TABLESPACE_NAME,
 NUM_ROWS,
 BLOCKS,
 EMPTY_BLOCKS,
 AVG_ROW_LEN
from user_tables;
ttitle off;

ttitle center 'Tables Defaults' skip 1 -
       center '===============' skip 2;
select
 TABLE_NAME,
 DEGREE,
```

```
 INSTANCES,
 INITIAL_EXTENT,
 NEXT_EXTENT
from user_tables;
ttitle off;
```

B.2.10 Oracle report on indexes

```
spool index.lst
set pagesize 999;

column index_name heading 'Index' format A20;
column table_owner heading 'Owner' format A10;
column table_name heading 'Table' format A18;
column table_type heading 'TableType' format A18;
column tablespace_name heading 'TableSpace' format A12;
column Uniqueness heading 'Unique' format A10;
column Blevel heading 'Blevel' format 999;
column Leaf_blocks heading 'Leaf|blocks' format 99999999;
column distinct_keys heading 'Distinct|Keys' format 9999999999;
column Status heading 'Status' format A8;
column column_name heading 'Column' format A18;
column column_position heading 'Position' format 999;
column column_length heading 'Length' format 999;

ttitle center 'Indexes - Info' skip 1 -
       center '==============' skip 2;
select INDEX_NAME,
--TABLE_OWNER,
TABLE_NAME,
TABLE_TYPE,
TABLESPACE_NAME
from user_indexes
order by TABLE_NAME;

ttitle center 'Indexes - Sizes' skip 1 -
       center '===============' skip 2;
select INDEX_NAME,
BLEVEL,
UNIQUENESS,
LEAF_BLOCKS,
DISTINCT_KEYS,
STATUS
from user_indexes
order by TABLE_NAME;

ttitle center 'Indexes - Columns' skip 1 -
```

```
       center '=================' skip 2;
select
 INDEX_NAME,
 TABLE_NAME,
 COLUMN_NAME,
 COLUMN_POSITION,
 COLUMN_LENGTH
from user_ind_columns
order by TABLE_NAME,index_name,COLUMN_POSITION;
ttitle off;
```

B.2.11 Oracle report on database files

```
spool file.lst
set pagesize 999;

ttitle center 'Database Files' skip 1 -
       center '==============' skip 2;
column tablespace_name heading 'TableSpace' format A12;
column file_name heading 'File' format A30;
column bytes heading 'Bytes' format 999,999,999,999;
column blocks heading 'Blocks' format 999,999,999;
column status heading 'Status' format A12;

select
 TABLESPACE_NAME, FILE_NAME, BYTES, STATUS
  FROM DBA_DATA_FILES
  ORDER BY TABLESPACE_NAME;
ttitle off;
```

B.2.12 Oracle report on extents

```
spool extents.lst
set pagesize 999;

ttitle center 'Tablespace Extents' skip 1 -
       center '==================' skip 2;
column tablespace_name heading 'TableSpace' format A12;
column segment_name heading 'Segment|Name' format A20;
column segment_type heading 'Seg.|Type' format A9;
column extent_id heading 'ID' format 999;
column bytes heading 'Bytes' format 9999999999;
column blocks heading 'Blocks' format 999999;

select
 TABLESPACE_NAME,
 SEGMENT_NAME,
```

```
 SEGMENT_TYPE,
 EXTENT_ID,
 BYTES,
 BLOCKS
from user_extents
order by TABLESPACE_NAME;
ttitle off;
```

B.2.13 Oracle report on parameters

```
spool parameter.lst
set pagesize 999;

ttitle center 'Non-Default Parameters' skip 1 -
       center '======================' skip 2;
col num format 9999 head 'Param#'
col name format a45 head 'Name' wrap
col type form 9999 head 'Type'
col value form a45 wrap head 'Value'
col isdefault form a9 head 'Default?'

select name, value
from v$parameter where isdefault = 'FALSE'
order by name;
ttitle off;
```

B.2.14 Oracle report on free space

```
spool free.lst
set pagesize 999;

ttitle center 'Free Space' skip 1 -
       center '==========' skip 2;
column tablespace_name heading 'TableSpace' format A12;
column file_id heading 'File|id' format 999999;
column block_id heading 'Block|id' format 999999;
column bytes heading 'Bytes' format 999,999,999,999;
column sum(bytes) heading 'Bytes' format 999,999,999,999;
column blocks heading 'Blocks' format 999,999,999;
column sum(blocks) heading 'Blocks' format 999,999,999;

select
 TABLESPACE_NAME,
 sum(BYTES),
 sum(BLOCKS)
from user_free_space
group by TABLESPACE_NAME;
```

```
ttitle off;

rem Full details but could be a large output
select
TABLESPACE_NAME,
FILE_ID,
BLOCK_ID,
BYTES,
BLOCKS
from user_free_space;
```

Appendix C. Reference sheets

The sections in this chapter contain reference sheets for standard SQL syntax, as well as for specific Oracle and DB2 UDB database commands. These reference sheets are not meant to serve as definitive references on their particular subject areas, but merely provide an overview of the use and syntax of some of the most commonly used SQL statements and database administration commands.

C.1 SQL reference sheet

This is a reference sheet for Structure Query Language (SQL). It contains numerous examples of SQL constructs and syntax with comments to explain what each SQL command does. SQL is a fairly straight forward, simple and readable language but still immensely powerful.

Rather than including dozens of pages in this redbook as a basic introduction to SQL, we have opted for this concise section to act as a reference and a reminder of SQL syntax. If you need an introduction to or advanced material on SQL, please refer to the section “Other resources” on page 415 for some excellent references to books on the subject.

These examples are based on the following sample tables:

```
create table emp(
      ename varchar2(30),
      job varchar2(20),
      loc number(10),
      deptno number(5),
      salary number(12)
      );

create table dept(
      deptno(5),
      dname varchar2(25),
      );
```

C.1.1 Data Definition Language (DDL) commands

Data Definition Language commands allow you to create, alter and delete database objects. They also allow you to grant and revoke certain database level authorities and privileges.

C.1.1.1 CREATE TABLE

Creates a database table consisting of one or more columns to contain user data.

```
create table anytable (col1 char(8) not null, col2 number(10), col3 date);
```

- `char` = Datatype
- `not null` = Constraints
- `(8)` = Length

```
create table localemp as select * from emp where deptno = 50;
```

`Create table as select` : results from the select clause are used to create the new table.

Data types

Char: character values

Varchar2: same as *char* without padding out of defined column length with blanks

Number: numerical values

Date: date values

Constraints

Integrity constraints restrict the value for one or more columns in a table.

Not Null - after a column definition specifies that the column must have a value for every row.

Primary Key - one column per table that must have a unique non-null value for that row.

C.1.1.2 CREATE VIEW

Creates alternate view of selected data or stores a complex query, which can be queried with a select statement.

```
create view bigbucks as select ename employee, job, sal from emp
where sal >= 3000;
```

C.1.1.3 CREATE INDEX

Indexes are used to speed up access to rows in a table. They are created on one or more columns of a table. The RDBMS decides when to use the index based on internal optimization techniques.

Primary Key: `create index empidx on emp(empno);`

Foreign key: `create index empdeptno on emp(deptno);`

Two Part Key: `create index emp2idx on emp(ename, sal);`

C.1.1.4 DROP

Used to delete an object from the database, such as a table, index or view.

```
drop view bigbucks;
drop table emp;
drop index emp_idx;
```

C.1.2 Data Manipulation Language (DML) commands

Data Manipulation Language commands are used to query and manipulate data contained in the database tables.

C.1.2.1 SELECT

Used to retrieve data from a table, view, or snapshot.

`select * from emp;` : retrieves all columns of data from the `EMP` table
`select ename, job from emp;` : retrieves only selected columns ename, job
`select ename "Employee", job from emp;` : provides *alias* name for column name
`select distinct job from emp;` : retrieves only distinct values (no duplicates)

WHERE clause

Used to restrict the rows returned in a query based on specific criteria. The `WHERE` clause is used in SQL `SELECT`, `INSERT`, `UPDATE`, and `DELETE` statements. Compound `WHERE` clauses are formed by using logical operators between clauses.

```
select deptno, dname, loc from dept where loc = 'Dallas';
```

Sub-queries

Allows a where clause to use a query result as criteria.

```
select ename, job from emp where deptno = (select deptno from emp where
ename = 'Smith');
```

Join query

Allows columns to be joined together from two or more tables in a query.

```
select ename, job, loc from emp, dept where emp.deptno = dept.deptno;
select ename, job, loc from dept, emp where dept.deptno = emp.deptno and
dname = 'RESEARCH';
```

ORDER BY clause
Used to order or sort the data returned by the query.

`select ename, job from emp order by sal;` : sorts rows by `sal` in ascending order

`select ename, job, deptno from emp order by deptno, sal desc;` : sorts by `depno` and `sal` in descending order

GROUP BY clause
Used to group selected rows and return summary information.

`select job, avg(sal) from emp group by job;` : returns average salary per job

HAVING clause
Used to restrict the groups returned by the `GROUP BY` clause.

```
select job, avg(sal) from emp group by job having avg(sal) < 2000;
```

C.1.2.2 INSERT
Used to add rows to a table.

```
insert into dept (deptno, dname, loc) values (50, 'IS', 'San Francisco');
```

Note: character strings or dates must be enclosed in single quotes.

`insert into dept values (50, 'IS', 'San Francisco');` : inserts values into all columns in correct order
`insert into newdept [col1, col2] select col1, col2 from dept [where ...];` : insert from other tables

C.1.2.3 DELETE
Used to remove rows from a table.

`delete from dept;` : deletes all rows in `dept` table
`delete from dept where deptno = 50;` : deletes rows based on where clause

C.1.2.4 UPDATE
Used to change existing column values in a table.

`update dept set dname = 'Networking';` : changes dname field in all rows
`update dept set dname = 'Networking' where deptno = 50;` : changes cols that meet where clause criteria

C.1.3 Operators
Used to manipulate individual data items within a query and return a result.

C.1.3.1 Relational operators

Used to compare one value within an expressions to another

Table 20. Relational operators.

=	equal to and
!=	not equal to
<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to
in	matches rows that have at least one value of a defined set `select * from abc where a in ('a','b');`
null	matches rows that have a null value (that is no value) in the column
not null	matches rows that do not have a null value in the column

like	character pattern matching - "_" (underscore) matches any single character per underscore.
	`select ename, job from emp where job like 'ANALY__'; (varchar2);`
	`select ename, job from emp where job like 'ANALY____'; (char(9));`

%	(percent sign) matches any number of characters at or after its position in the value. `select ename, job from emp where job like 'AN%';`

C.1.3.2 Logical operators

Used to create a compound where clauses using AND or OR

```
select ename, job from emp where job = 'CLERK' and hiredate > '03-DEC-81';
select ename, job from emp where job = 'CLERK' or job like '%MAN%';
```

NOT - negates conditional expression, also used in single expressions

```
select ename, job from emp where sal >= 3000 and NOT job = 'PRESIDENT';
```

C.1.3.3 ANY or ALL operators

Used when sub-queries return more than one row.

- ANY returns any true result from the sub-query back to the main query.
- ALL returns results back to the main query only when all the sub-query results will cause the where clause to evaluate as true.

C.1.3.4 Set operators

Used to combine the results of two component queries into a single result.

`UNION` : merges two result sets into one combined set

`INTERSECT` : selects just the common values of two result sets

`MINUS` : selects values in the first result set not present in the second

C.1.3.5 Expression operators

Used to perform an operation on the underlying data before returning the result.

Arithmetic: + (plus), - (minus), * (multiply), / (divide)

Character: || (concatenate)

```
select firstname | ' ' | lastname from .....
```

C.1.4 SQL functions

These SQL functions are used to modify or format the data that is returned by SQL statements. They are used to make the data more readily understood so that the data (in the new format) can be used in the SQL `WHERE` clause.

C.1.4.1 Aggregate group functions

Used to select on columns to provide group summary information.

`count(sal)`: returns count of non-null rows (numeric, character or date)

`count(*)`: returns count of all rows (numeric, character or date)

`min(sal)`: returns minimum value (numeric, character or date)

`max(sal)`: returns maximum value (numeric, character or date)

`avg(sal)`: returns average value (numeric only)

`sum(sal)`: returns sum of rows (numeric only)

C.1.4.2 Common functions (Oracle only)

These are shorthands to use in the following section:

- `f` = function
- `c` = char values or column
- `n` = numeric values

Single row character functions

`initcap - f(c)`: capitalizes the first letter of a word leaving the rest lowercase

`upper - f(c)`: changes a string to all UPPERCASE letters

`lower - f(c)`: changes a string to all lowercase letters

`length - f(c)`: returns the character length of a string

`lpad - f(c1,n,c2)`: pads c1 with n characters to the left. c2=pad character

`rpad - f(c1,n,c2)`: pads c2 with n characters to the right. c2=pad character

`substr - f(c,n1,n2)`: extracts a sub-string starting at n1. n2=substring length

`instr - f(c1,c2,n1,n2)`: returns position of c2 in c1, n1=starting position in c1 n2=occurrence of c2

`decode - f(expr,c1,trans1,c2,trans2,...,default)`: evaluates expression and substitutes 'translation' if 'c' is present. 'default' returned if no match.

```
decode(salecode, 1, 'books', 2, 'toys', 'misc');
```

Single row number functions

Used to manipulate numeric input.

`round - round(n1,n2)`: rounds n1 to n2 precision of decimal places

`trunc - trunc(n1,n2)`: truncates n1 to n2 decimal places

`abs - abs(n)`: returns the absolute value of n

`nvl - nvl(expr1,expr2)`: if a value is null then `expr2` is substituted for `expr1`. If not, `expr1` is used. `expr1` must be the same datatype as `expr2`.

Conversion functions

Used to convert data types from one type to another.

`to_char - to_char(expr, format mask)`: `to_char(hiredate, 'MM/DD/YY')` will change standard DD-MON-YY output to "10/07/93" style.

`to_date - to_date(chardate, format mask)`: `to_date('10/07/93','MM/DD/YY')` will convert to a date datatype that can be stored in the database.

`to_number - f(c)`: converts a `char` numeric value to numeric.

Date functions
Used to operate on values of datatype *DATE*.

`add_months(date, n) - add_months(sysdate, +2)`: returns current month + 2

`last_day(date) - last_day('07-OCT-93')`: returns last day in date's month

`next_day(date, day) - next_day('07-OCT-93', 'Sat')`: returns date of next day after date

`months_between(date1, date2)`: returns number of months between two dates

C.2 Oracle SQLplus extensions reference sheet

This section is a reminder of how to use one of the most useful additions to SQL available within Oracle: the interactive monitor called `sqlplus` (sometimes printed `SQL*Plus`).

C.2.1 Running files and editing

- `@filename.sql` runs file containing SQL commands
- `run` runs the SQL commands currently in the buffer
- `ed` starts the editor (`vi` by default) on the current buffer
- parameters: using `&number` in a SQL file for parameter substitution when using *start* to run the script)

  ```
  select * from emp where mgr = &1
  ```

  then use: `@myscript 7698` and sqlplus will substitute the first parameter

C.2.2 Line editing commands

Most people prefer using the editor (use `ed` command) provided you are familiar with the UNIX editor `vi`. This will work with SQL commands only. SQL*PLUS commands are not captured in buffer.

- `list (l)`: list the current SQL commands in the buffer. The current line is marked with an asterisk.
- `change (c)`: `c /old/new` - changes current line. `- c /typo/` deletes the string *typo* from current line.
- `append (a)`: `a from emp` - appends to end of current line.
- `input (i)`: add more command lines after the current line. Hit return on a line by itself to end.
- `del`: delete the current line from the buffer.

- `n "text"`: replaces text on the command line "`n`".
- `clear buffer (cl buff)`: clears contents of SQL command buffer.

C.2.3 Report/formatting commands

- `describe (object)`: describes columns, data types in order defined.
  `desc emp` - describe the `emp` column
- `column`: formats column display with heading and format specification
  ```
  column loc format a8 heading 'City'
  column sal format 9999 heading 'Bucks'
  ```
- `ttitle`: provides report title information at the top of each page
  ```
  ttitle [left|center|right] 'text' skip [left|center|right] 'text'
  ttitle 'My Report on EMP' skip '=============='
  ```
  Note: `skip` forces `'text'` to skip to next line
- `btitle`: same syntax as `ttitle` - provides report title at bottom of page
- `break`: defines a control break where values in a column change from the previous row - used to perform some formatting actions.
  Note: for the output to be meaningful, the column should be sorted.
  ```
  break on {expr|col} [skip n| skip page] [duplicates|nodup]
  break on job skip 2
  select job, ename from emp order by job;
  ```
  Will skip 2 lines between jobs & not show duplicates (default)
- `compute`: performs a calculation at a defined control break. Standard group functions can be used for the calculation type
  ```
  compute count of ename on job
  select job, ename from emp order by job;
  ```
  Performs the count calculation when a break is detected for the job column. The result is printed on a separate line before the break *skip*.

C.2.4 Miscellaneous

`set pause {on | off};` : Set on to stop scrolling returned rows off the screen.

`set linesize N;` : Sets the width of the screen to stop wrapping space filled. - set linesize 200;

`set pagesize N;` : Sets how often column headings are output (default 24) use with spool.

`set prompt { on | off | 'string' };` : Sets the next command prompt

`set time { on | off };` : The prompt now includes the time.

`set timing { on | off };` : After each SQL statement the time taken is reported

`set termout { on | off };` : Stops the output to the screen (assuming you used spool already)

`spool [filename] {on | off};` : Toggles capturing SQL*PLUS output to a file. Default .lst file.

```
spool mysession
....
spool off
```

`host <ls>" or "!<dir>";` : Execute the host operating system command.

`exit` or `^D` to stop

C.2.5 Help and additional settings

Enter `Help commands` or `help menu` for an overview of SQL, SQL*PLUS and PL/SQL topics.

Setting option for all SQL*PLUS sessions:

```
$ define _EDITOR=<host_editor_name>
```

login.sql = file with commands to execute when SQL*PLUS starts

C.3 Oracle DBA reference sheet

This section contains selected Oracle DBA commands. Please consult one of the references listed in Appendix F.3, “Other resources” on page 415 for more definitive explanations of the commands listed in this section.

C.3.1 Storage-Clause

This clause is used in many places. For an example see the `CREATE TABLESPACE` section.

```
(INITIAL integer [K|M]      | NEXT  integer [K|M]
| MINEXTENTS integer     | MAXEXTENTS integer
| PCTINCREASE integer
| OPTIMAL [NULL | integer [K|M] ]  -- rollback only
| FREELISTS integer -- tables and indexes only
|FREELIST GROUPS integer -- tables only
)
```

C.3.2 ALTER DATABASE

To alter an existing database in one of these ways:

- mount the database
- open the database
- archivelog or noarchivelog mode for redo log groups

```
ALTER DATABASE mydb MOUNT EXCLUSIVE;
ALTER DATABASE mydb OPEN;
ALTER DATABASE mydb ARCHIVELOG;
ALTER DATABASE mydb NOARCHIVELOG;
```

C.3.3 ALTER INDEX

To change future storage allocation for data blocks in an index.

```
ALTER INDEX empidx STORAGE storage_clause;
```

C.3.4 ALTER ROLLBACK SEGMENT

To alter a rollback segment in one of these ways:

- by bringing it online
- by taking it offline
- by changing its storage characteristics

```
ALTER ROLLBACK SEGMENT rollback1 ONLINE;
ALTER ROLLBACK SEGMENT rollback1 OFFLINE;
ALTER ROLLBACK SEGMENT roll1 STORAGE storage_clause;
```

C.3.5 ALTER SESSION

To alter your current session in one of these ways:

- to enable or disable the SQL trace facility
- to change the goal of the cost-based optimization approach

```
ALTER SESSION SET SQL_TRACE  = TRUE;
ALTER SESSION SET SQL_TRACE  = FALSE;
ALTER SESSION NLS_DATE_FORMAT   =  'fmt';
ALTER SESSION OPTIMIZER_GOAL   = RULE;
possible GOALs are RULE, ALL_ROWS, FIRST_ROWS or CHOOSE
```

C.3.6 ALTER SYSTEM

To dynamically alter your Oracle instance in one of these ways:

- to explicitly switch redo log file groups

- to explicitly perform a checkpoint
- to manually archive redo log file groups or to enable or disable automatic archiving

```
ALTER SYSTEM CHECKPOINT;
ALTER SYSTEM SWITCH LOGFILE;
ALTER SYSTEM ARCHIVE LOG STOP;
ALTER SYSTEM ARCHIVE LOG START;
```

C.3.7 ALTER TABLE

To alter the definition of a table in one of these ways:

- to add a column
- to modify storage characteristics or other parameters
- to explicitly allocate an extent

```
ALTER TABLE emp ADD column ( newcol varchar2(10) );
ALTER TABLE emp PCTFREE 10 PCTUSED 60 [STORAGE storage_clause];
ALTER TABLE emp ALLOCATE EXTENT SIZE 200K DATAFILE '/dev/lv_emp3';
ALTER TABLE table PARALLEL ( DEGREE 8);
```

C.3.8 ALTER TABLESPACE

To alter an existing tablespace in one of these ways:

- to add data file(s)
- to change default storage parameters
- to take the tablespace online or offline

```
ALTER TABLESPACE myts ADD DATAFILE (filespec [, filespec] ... );
ALTER TABLESPACE myts DEFAULT STORAGE (storage_clause);
ALTER TABLESPACE myts ONLINE;
ALTER TABLESPACE myts OFFLINE;
ALTER TABLESPACE myts BEGIN BACKUP;
ALTER TABLESPACE myts END BACKUP;
```

14.5.1 ALTER USER

To change any of these characteristics of a database user:

- password
- default tablespace for object creation
- tablespace for temporary segments created for the user

```
ALTER USER fred IDENTIFIED BY funny DEFAULT TABLESPACE usr;
TEMPORARY TABLESPACE bigsort;
```

```
ALTER USER fred IDENTIFIED EXTERNALLY;
```

C.3.9 ANALYZE

Validate the index, table and collect the cost based optimizer statistics.

```
ANALYZE TABLE emp COMPUTE STATISTICS;
ANALYZE TABLE emp ESTIMATE STATISTICS SAMPLE 1000 ROWS;
ANALYZE TABLE emp ESTIMATE STATISTICS SAMPLE 10 PERCENT;
ANALYZE INDEX empidx COMPUTE STATISTICS;
```

C.3.10 CREATE DATABASE

To create a database, making it available for general use, with these options:

- to establish a maximum number of instances, data files, redo log files groups, or redo log file members
- to specify names and sizes of data files and redo log files
- to choose a mode of use for the redo log

Attention

This command prepares a database for initial use and erases any data currently in the specified files. Only use this command when you understand its ramifications.

```
CREATE DATABASE model CONTROLFILE REUSE
   DATAFILE '/dev/rsystem'  SIZE 255M
   LOGFILE  '/dev/rredo1'   SIZE 255M,'/dev/rredo2'   SIZE 255M
   MAXINSTANCES 1
   MAXLOGFILES  255
   MAXDATAFILES 1022
   ARCHIVELOG
   CHARACTER SET "US7ASCII";
```

C.3.11 CREATE INDEX

To create an index on one or more columns of a table or a cluster. An index is a database object that contains an entry for each value that appears in the indexed column(s) of the table and provides direct, fast access to rows.

```
CREATE INDEX empidx ON emp (empno)
   TABLESPACE index_ts
   STORAGE storage_clause]
   PARALLEL ( DEGREE 8);
```

C.3.12 CREATE ROLLBACK SEGMENT

To create a rollback segment. A rollback segment is an object that is used by Oracle to store data necessary to reverse, or undo, changes made by transactions.

```
CREATE ROLLBACK SEGMENT roll1 TABLESPACE rb
   STORAGE (
   INITIAL        2M
   NEXT           2M
   MINEXTENTS     10
   MAXEXTENTS     249
   OPTIMAL        20M
   );
```

C.3.13 CREATE TABLE

To create a table, the basic structure to hold user data, specifying this information:

- column definitions
- integrity constraints
- the table's tablespace
- storage characteristics
- data from an arbitrary query

```
CREATE TABLE ORDERS (O_ORDERKEY       INTEGER NOT NULL,
      O_CUSTKEY        INTEGER NOT NULL,
      O_ORDERSTATUS    CHAR(1) NOT NULL,
      O_TOTALPRICE     FLOAT NOT NULL,
      O_ORDERDATE      DATE NOT NULL,
      O_ORDERPRIORITY  CHAR(15) NOT NULL,
      O_CLERK          CHAR(15) NOT NULL,
      O_SHIPPRIORITY   INTEGER NOT NULL,
      O_COMMENT        VARCHAR(79) NOT NULL)
   TABLESPACE other
   STORAGE ( INITIAL 100M
      NEXT 100M
      PCTINCREASE 0
      FREELISTS 40
      FREELIST GROUPS 2)
   PARALLEL (DEGREE 8);

CREATE TABLE new_emp AS select * from emp;

CREATE TABLE emp2 (
      empno number(6),
```

```
      deptno number(5),
      ename varchar2(30),
   salary number(10)) CONSTRAINT uni1 UNIQUE (empno);

CREATE TABLE emp2 (
      empno number(6),
      deptno number(5),
      ename varchar2(30),
      salary number(10))
   CONSTRAINT emp_pk PRIMARY KEY (empno)
   CONSTRAINT emp_fk1 FOREIGN KEY (deptno) REFERENCES dept (deptno)
   USING INDEX TABLESPACE indexts;
```

C.3.14 CREATE TABLESPACE

To create a tablespace. A tablespace is an allocation of space in the database that can contain objects.

```
CREATE TABLESPACE empts DATAFILE '/dev/rdata1' SIZE 1023M REUSE
   DEFAULT STORAGE (
      INITIAL      100M
      NEXT         100M
      PCTINCREASE  0
      MINEXTENTS   1
      MAXEXTENTS   249
   );
```

C.3.15 CREATE USER

To create a database user, or an account through which you can login to the database, and establish the means by which Oracle permits access to the database by the user.

```
CREATE USER fred IDENTIFIED BY funny DEFAULT TABLESPACE usr;
TEMPORARY TABLESPACE bigsort;
CREATE USER fred IDENTIFIED EXTERNALLY;
```

C.3.16 CREATE VIEW

To define a view, a logical table based on one or more tables or views.

```
CREATE VIEW view AS subquery;
```

C.3.17 DROP

To delete an object within the database.

```
DROP INDEX index;
DROP ROLLBACK SEGMENT rollback_segment;
DROP TABLE table [CASCADE CONSTRAINTS];
```

```
DROP TABLESPACE tablespace [INCLUDING CONTENTS [CASCADE CONSTRAINTS]] ;
DROP USER user [CASCADE];
DROP VIEW view;
```

C.3.18 EXPLAIN PLAN

To determine the execution plan Oracle follows to execute a specified SQL statement. This command inserts a row describing each step of the execution plan into a specified table. If you are using cost-based optimization, this command also determines the cost of executing the statement.

```
EXPLAIN PLAN;
SET STATEMENT ID = 'myplan'];
FOR SQLstatement;
```

C.3.19 RENAME

To rename a table, view, sequence, or private synonym.

```
RENAME old TO new;
```

C.3.20 TRUNCATE

To remove all rows from a table or cluster.

```
TRUNCATE TABLE table;
TRUNCATE TABLE table DROP STORAGE;
```

C.3.21 Useful Oracle internal tables

```
DBA_CONSTRAINTS
DBA_CONS_COLUMNS
DBA_DATA_FILES
DBA_EXTENTS
DBA_FREE_SPACE
DBA_INDEXES
DBA_IND_COLUMNS
DBA_ROLLBACK_SEGS
DBA_SEQUENCES
DBA_TABLES
DBA_TABLESPACES
DBA_TAB_COLUMNS
DBA_USERS
DBA_VIEWS

USER_CONSTRAINTS
USER_CONS_COLUMNS
USER_EXTENTS
USER_FREE_SPACE
```

```
USER_INDEXES
USER_IND_COLUMNS
USER_SEQUENCES
USER_TABLES
USER_TABLESPACES
USER_TAB_COLUMNS
USER_VIEWS
```

Use `sqlplus describe` to find out the details of these tables:

```
sqlplus> desc user_tables;
```

C.4 DB2 UDB DBA reference sheet

This section contains selected DB2 UDB DBA commands. Please consult one of the references listed in Appendix F.3, “Other resources” on page 415 for more definitive explanations of the commands listed in this section.

C.4.1 ALTER BUFFERPOOL

To alter the buffer pool in one of the following ways:

- modify the size of the buffer pool on all partitions (or nodes) or on a single partition
- turn on or off the use of extended storage
- add this buffer pool definition to a new nodegroup.

```
ALTER BUFFERPOOL bufferpoolname;
ALTER BUFFERPOOL bufferpoolname NOT EXTENDED STORAGE;
ALTER BUFFERPOOL bufferpoolname ADD NODEGROUP nodegroupname;
```

C.4.2 ALTER TABLE

To alter an existing table in one of the following ways:

- Adding one or more columns to a table
- Adding or dropping a primary key
- Adding or dropping one or more unique or referential constraints
- Altering the length of a VARCHAR column
- Altering a reference type column to add a scope
- Adding or dropping a partitioning key
- Setting the table to `not logged initially` state

```
ALTER TABLE emp ADD column ( newcol varchar2(10) );
```

```
ALTER TABLE emp ALTER column SET DATA TYPE VARCHAR (integer);
ALTER TABLE emp ADD PARTITIONING KEY (column-name);
ALTER TABLE emp DROP CONSTRAINT (contraint-name);
```

C.4.3 ALTER TABLESPACE

To alter an existing tablespace in one of these ways:

- Add a container to a DMS table space (that is, one created with the MANAGED BY DATABASE option).
- Add a container to a SMS tablespace on a partition (or node) that currently has no containers.
- Modify the PREFETCHSIZE setting for a tablespace.
- Modify the BUFFERPOOL used for tables in the tablespace.
- Modify the OVERHEAD setting for a tablespace.
- Modify the TRANSFERRATE setting for a tablespace.

```
ALTER TABLESPACE myts ADD (DEVICE '/dev/rhdisk9' 10000);
ALTER TABLESPACE myts PREFETCHSIZE 64 OVERHEAD 19.3;
ALTER TABLESPACE myts BUFFERPOOL bufferpoolname;
ALTER TABLESPACE myts SWITCH ONLINE;
ALTER TABLESPACE myts OVERHEAD milliseconds;
```

C.4.4 CREATE DATABASE

Initializes a new database with an optional user-defined collating sequence,creates the three initial table spaces, creates the system tables, and allocates the recovery log.

```
CREATE DATABASE model;
```

C.4.5 CREATE INDEX

To create an index on one or more columns of a table or a cluster. An index is a database object that contains an entry for each value that appears in the indexed column(s) of the table and provides direct, fast access to rows.

```
CREATE INDEX empidx ON emp (empno);
```

C.4.6 CREATE TABLE

To create a table, the basic structure to hold user data, specifying this information:

- column definitions
- integrity constraints

- the table's tablespace
- storage characteristics
- data from an arbitrary query

```
CREATE TABLE ORDERS ( O_ORDERKEY       INTEGER NOT NULL,
     O_CUSTKEY        INTEGER NOT NULL,
     O_ORDERSTATUS    CHAR(1) NOT NULL,
     O_TOTALPRICE     FLOAT NOT NULL,
     O_ORDERDATE      DATE NOT NULL,
     O_ORDERPRIORITY  CHAR(15) NOT NULL,
     O_CLERK          CHAR(15) NOT NULL,
     O_SHIPPRIORITY   INTEGER NOT NULL,
     O_COMMENT        VARCHAR(79) NOT NULL)
     in TSMED
     INDEX in TSMED;

CREATE TABLE new_emp AS select * from emp;

CREATE TABLE emp2 (
      empno number(6),
      deptno number(5),
      ename varchar2(30),
   salary number(10)) CONSTRAINT uni1 UNIQUE (empno);

CREATE TABLE emp2 (
      empno number(6),
      deptno number(5),
      ename varchar2(30),
      salary number(10))
   CONSTRAINT emp_pk PRIMARY KEY (empno)
   CONSTRAINT emp_fk1 FOREIGN KEY (deptno) REFERENCES dept (deptno)
   INDEX IN indexts;
```

C.4.7 CREATE TABLESPACE

To create a new tablespace within the database, assigns containers to the tablespace, and records the tablespace definition and attributes in the catalog.

```
CREATE TABLESPACE empts MANAGED BY SYSTEM
 USING ('/DB2DATA/FSEMP10/CONT1',
        '/DB2DATA/FSEMP11/CONT2',
        '/DB2DATA/FSEMP12/CONT3',
        '/DB2DATA/FSEMP13/CONT4')
         EXTENTSIZE 32
         PREFETCHSIZE 128;
```

C.4.8 CREATE VIEW

To define a VIEW statement that creates a view on one or more tables, views or nicknames.

```
CREATE VIEW view AS subquery;
```

C.4.9 DROP

To delete an object within the database.

```
DROP ALIAS alias;
DROP INDEX index;
DROP TABLE table;
DROP TABLESPACE tablespace;
DROP VIEW view;
```

C.4.10 EXPLAIN PLAN

The EXPLAIN statement captures information about the access plan chosen for the supplied explainable statement and places this information into the Explain tables. An explainable statement is a DELETE, INSERT, SELECT, SELECT INTO, UPDATE, VALUES, or VALUES INTO SQL statement.

```
EXPLAIN PLAN
SET QUERYNO = integer
FOR SQLstatement;
```

C.4.11 RENAME TABLE

To rename an existing table.

```
RENAME oldname TO newname;
```

C.4.12 Useful DB2 UDB internal catalog views

```
SYSCAT.TABAUTH
SYSCAT.TABLES
SYSCAT.TABLESPACES
SYSCAT.INDEXES
SYSCAT.COLUMNS
SYSCAT.VIEWS
SYSSTAT.COLUMNS
SYSSTAT.INDEXES
SYSSTAT.TABLES
```

Appendix D. The Model Database used for testing in this redbook

For the purposes of this redbook we used a database that is refered to as the Model Database.This database's schema, tables, indexes, tools and application are described in this appendix. The Model Database is based on the well known database used within the computer industry standard benchmark from the Transaction Processing Performance Council[1] (TPC) for decision support systems. In 1999 the TPC created two decision support benchmarks (based on the older TPC-D) called TPC-H (for ad-Hoc queries) and TPC-R (for Reporting queries). For more information on these benchmarks please refer to URL:

```
http://www.tpc.org
```

The Model Database is based on the database defined in these benchmarks. But note that our Model Database does not have a standard size for reporting TPC results, not all the queries were used and the results were not audited. All of these are mandatory for official TPC results.

Test results

Results from this exercise cannot be compared to any official TPC results.

This Model Database is used purely to compare the relative performance of various RS/6000 hardware, AIX operating system and database configuration options.

This particular database is used because it is:

- well known to many people
- fairly typical of an order processing system
- simple enough to be quickly build, the data generated, the data loaded and then indexed
- available with a database data generator that can create any size of database from one MB to GBs and even up to a TB
- suitable for very complex decision support queries

To this standard database we have added:

- a typical set of indexes
- an application that can generate OLTP type workloads of various SQL statements and that reports on the transaction rates achieved
- scripts to create the disk space, tablespaces, tables, indexes and statistics

[1] TPC, TPC-D, TPC-H and TPC-R are copyrights of the Transaction Processing Performance Council.

- monitoring tools for analyzing the database and AIX performance

The result is a practical and useful database for investigating performance issues and options. It is also excellent for skills development.

D.1 Schema

The database has eight objects (tables) and models a typical but simple order processing system with customers, parts, supplier and sales details. This schema is sufficiently complex and life like to make any of the SQL statements that you find in a production system available within this test database. The following diagram shows the tables and relationships between tables. The sizes of the boxes try to highlight the size of the tables (but are not strictly to scale) and the arrows show the one to many relationships.

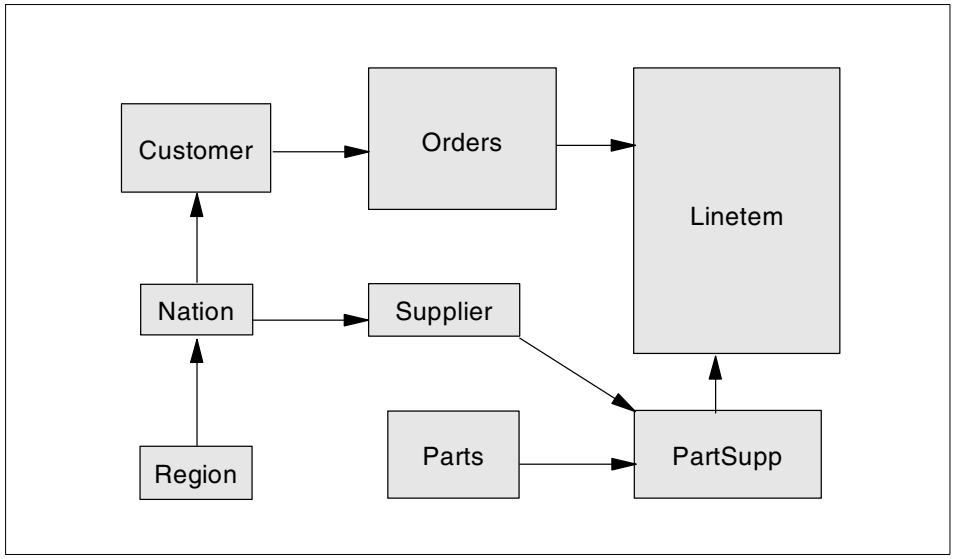


Figure 44. The Model Database Schema

D.2 The model database tables

The tables of the model database are straight forward and detailed below. The size scales depending on the size of the generated database. The below assumes a data size of 1 GB .

Table 21. Model database tables and their sizes

Table	Row size	Rows	Size in Mbytes	Percentage of database
supplier	147	10,000	1.4	0.1
parts	133	200,000	25.37	2.4
partsupp	146	800,000	111.39	10.6
customer	162	150,000	23.17	2.2
orders	113	1,500,000	161.65	15.3
lineitem	128	6,000000	732.57	69.4
nation [1]	109	25	0	0.0
region [1]	116	5	0	0.0
Totals		8,661,245	1055.55	100
Note 1 These tables do not scale for larger database sizes.				

From this you can see the majority of the database data volume is contained in the *lineitem* table. On average there are six lineitem rows for each row in the orders table. Note that the orders table has plural. This is to avoid possible confusion as some vendors use the word *order* as a keyword in the `create table` command. To model a real production database, it is necessary to have some tables that are much larger than the available memory, so that there is no possibility that the complete table can be loaded into the shared memory cache of the database. The large size of the *orders* and *lineitem* tables does this, as they are a large proportion of the total database data volume.

D.3 The model database indexes

A typical set of database indexes were created for the database based on:

- primary keys
- foreign keys

- typical look up requirements like: customer/supplier name and dates for date range queries.

The indexes used are created as below:

Primary keys

```
CREATE INDEX pk_p_partkey ON parts (p_partkey)
CREATE INDEX pk_s_suppkey ON supplier (s_suppkey)
CREATE INDEX pk_ps_partsupp ON partsupp (ps_partkey,ps_suppkey)
CREATE INDEX pk_c_custkey ON customer (c_custkey)
CREATE INDEX pk_o_orderkey ON orders (o_orderkey)
CREATE INDEX pk_l_okeyline ON lineitem (l_orderkey,l_linenumber)
```

The nation and region tables are too small to require indexes

Foreign keys

```
CREATE INDEX fk_o_ckey ON orders (o_custkey)
CREATE INDEX fk_l_pkey ON lineitem (l_partkey)
CREATE INDEX fk_l_skey ON lineitem (l_suppkey)
```

Useful date fields as they are often used in the SQL

```
CREATE INDEX o_odate ON orders (o_orderdate)
CREATE INDEX l_sdate ON lineitem (l_shipdate)
CREATE INDEX l_rdate ON lineitem (l_receiptdate)
CREATE INDEX l_cdate ON lineitem (l_commitdate)
```

D.4 OLTP workload generation

The transaction generating tool, called `oltp`, can repeatedly start transactions. This tool can:

- Start transactions, one immediately after the other or with a user think time interval between transactions.
- Report to the user performance statistics on a regular basis including
  - run time so far
  - number of transaction finished so far
  - the transaction rate per second- so far
  - the transaction rate per second - recently
  - the minimum, average and maximum time the transaction took down to the nearest millisecond
- Stop after a fixed time

- Use a number of widely varying different transactions including
  - simple one row select
  - 2 table join select
  - 4 table join select
  - cursors
  - inserts
  - simple and complex updates
  - select with sub-query
  - complex transactions (multiple selects, inserts and updates)

To avoid the problem of deciding the rate at which users start transactions, the tool normally uses a continuously running mode (meaning with no user think time). As a result, this gives the transaction rate that the system can deliver. To achieve this transaction rate, more than the number of processors in the system simultaneous `oltp` programs must be run at the same time. This ensures that there is always a transaction being performed by the database at any one particular time. This OLTP transaction generating program is written in the C language with embedded SQL and is available for IBM internal use.

As a rough guide a user could initiate one transaction every 30 seconds. So for example, the user would be inputting data at the PC based application for 30 seconds and then request the database to save the data or alternatively the user requests data from the database and then takes 30 seconds to read and review the data before requesting more. This means the number of active users the system can support would be estimated as 30 times the transaction rates in seconds. For example, if the machine can do 100 transactions a second and the user would do a transaction every 30 seconds then the number of users it can support is:

100 transactions/second * 30 seconds/transaction = 3000 users

The `oltp` command interface is:

```
oltp [-d] -s <1-17> [-t <seconds>] -f <filename> -u <user/pass> [-r
seed] [-m max-seconds] [-i idle-seconds]
Version 10.1
 -s n = which sql to use (mandatory)
 -f f = file of valid orderkeys (mandatory)
 -u u/p = username and password (i.e. scott/tiger mandatory)
 -t s = seconds between reporting transaction rates (default 30)
 -h   = full    help info
 -?   = summary help info
```

```
-d   = verbose output
-r s = set random seed value
-m s = maximum seconds of run time before existing
```

The OLTP SQL transactions are as follows:

Name	Description
sql1	one row select (all columns) from the orders table
sql2	two tables join and select limited columns from the orders and lineitems tables
sql3	five tables join and select limited columns from orders, lineitems, parts, supplier and customer
sql4	use cursor to select master and multiple slave rows from orders and lineitems
sql5	use cursor to select master and multiple slave rows with an 'order by' from orders and lineitems
sql6	insert rows in the orders and lineitem tables
sql7	find the number of rows in each table
sql8	update one row in the orders table
sql9	update two rows in the orders table within one transaction
sql10	update the same row in the orders table over and over again
sql11	repeatedly update one row from a small set of rows in the orders table
sql13	select four columns from the customer table
sql14	select four columns from the supplier table
sql15	select min(value) from parts using a subquery
sql16	large transaction using sql13, sql14, sql15, sql8 sql16

Figure 45. OLTP transaction summary

D.5 DSS workload generation

Only four standard queries were used to make up the decision support system (DSS) test. These were a range of query types including:

- fairly simple SQL statements which would require scanning entire tables
- joining two very large tables with restrictions

- joining many very large tables with restrictions
- extremely complex queries involving sub-queries

Each SQL statement was run one at a time with an RBDMS restart between each test to eliminate the benefits of preloading the RDBMS disk cache so that each test run was consistent. In a production database significant performance gains would be made be if the RDBMS disk cache already contained the entire contents of smaller tables.

The complex queries used to generate decision support workloads are below:

D.5.1 Query 2

```
SELECT
    S_ACCTBAL,S_NAME,N_NAME,P_PARTKEY,P_MFGR, S_ADDRESS, S_PHONE, S_COMMENT
FROM PARTS, SUPPLIER, PARTSUPP, NATION, REGION
WHERE
    P_PARTKEY = PS_PARTKEY
    AND S_SUPPKEY = PS_SUPPKEY
    AND P_SIZE = 15
    AND P_TYPE LIKE '%BRASS'
    AND S_NATIONKEY = N_NATIONKEY
    AND N_REGIONKEY = R_REGIONKEY
    AND R_NAME = 'EUROPE'
    AND PS_SUPPLYCOST = ( SELECT
                                MIN(PS_SUPPLYCOST)
                            FROM PARTSUPP, SUPPLIER, NATION, REGION
                    WHERE
                                P_PARTKEY = PS_PARTKEY
                                AND S_SUPPKEY = PS_SUPPKEY
                                AND S_NATIONKEY = N_NATIONKEY
                                AND N_REGIONKEY = R_REGIONKEY
                                AND R_NAME = 'EUROPE')
ORDER BY
     S_ACCTBAL DESC, N_NAME,S_NAME,P_PARTKEY;
```

D.5.2 Query 6

```
SELECT SUM(L_EXTENDEDPRICE*L_DISCOUNT) AS REVENUE
FROM LINEITEM
WHERE
    L_SHIPDATE >= TO_DATE('1994-01-01')
    AND  L_SHIPDATE < ADD_MONTHS(TO_DATE('1994-01-01'),12)
    AND  L_DISCOUNT BETWEEN 0.06 - .01 AND 0.06 + .01
    AND  L_QUANTITY < 24;
```

D.5.3 Query 13

```
SELECT TO_CHAR(O_ORDERDATE,'YYYY'), SUM(L_EXTENDEDPRICE * (1-L_DISCOUNT))
FROM LINEITEM, ORDERS
WHERE
       O_ORDERKEY  =  L_ORDERKEY
       AND  O_CLERK  = 'Clerk#000000088'
       AND  L_RETURNFLAG = 'R'
GROUP BY TO_CHAR(O_ORDERDATE,'YYYY')
ORDER BY TO_CHAR(O_ORDERDATE,'YYYY');
```

D.5.4 Query 17

```
SELECT SUM(L_EXTENDEDPRICE)/7.0
FROM LINEITEM, PARTS
WHERE
   P_PARTKEY = L_PARTKEY
   AND   P_BRAND = 'Brand#23'
   AND   P_CONTAINER = 'MED BOX'
   AND   L_QUANTITY < ( SELECT
                        0.2*AVG(L_QUANTITY)
                        FROM LINEITEM
                        WHERE
                        L_PARTKEY = P_PARTKEY);
```

Test Results

Results from this exercise cannot be compared to any official TPC results.

This Model Database is used purely to compare the relative performance of various RS/6000 hardware, AIX operating system and database configuration options.

D.6 Model Database physical layout

For the purposes of the redbook a particular database size and disk layout was used in all the tests. The data was generated to provide 5 GB of raw data (five times the sizes in Table 21 on page 403). Sixteen disks were used for the database as follows:

Table 22. Physical layout

Use	Number of Disks	Approximate disk size GB
data	4	8 [1]
index	4	8

Use	Number of Disks	Approximate disk size GB
tmp/sort	4	8
RDBMS code	1	2
logs	1	2
redo	1	2
system/catalog	1	2
Totals	16	32
Note 1 5 GB of data was placed within 8 GB of disk space. This avoids any possible problems loading the data into the database due to database overhead in storing data and space limitations on the disks.		

The 5 GB of raw data size was chosen as the machines were 4 way SMP RS/6000s and all have 1 GB of memory. This database size means the RDBMS disk cache was typically 512 MB in size (based on the rule of thumb where 50% of memory is allocated to the shared memory). The raw data plus indexes were approximately 8 to 10 GB in size. Therefore the disk size to cache size ratio was between 15:1 to 20:1.This makes sure that in our tests the RDBMS will be forced to use the disks. This ratio is similar to many production systems.

The separation of data, index and tmp/sort areas is often found in production systems but there is also the approach of mixing these to even out the I/O workload across the maximum number of disks. However, in these tests the separation is useful because it becomes very simple to determine which part of the database is in use. For example, some SQL statements can be answered by only using the index (where all columns of interest are in the index). This can clearly be seen when monitoring which disks are in use.

There are many ways of creating the logical volumes/partitions with AIX within the above physical layout, including, RAID 5, striping and mirroring or straight logical volume per disk

Test Results

Results from this exercise cannot be compared to any official TPC results.

This Model Database is used purely to compare the relative performance of various RS/6000 hardware, AIX operating system and database configuration options.

Appendix E. Special notices

This publication is intended as a guideline for System Designers and Architects to design a well performing RDBMS on an IBM RS/6000. It also helps System Administrators and Database Administratorson to setup, use, tune and maintain an RDBMS for optimal performance. See the PUBLICATIONS section of this redbook 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, North Castle Drive, Armonk, NY 10504-1785.

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.

Any pointers in this publication to external Web sites are provided for convenience only and do not in any manner serve as an endorsement of these Web sites.

Any performance data contained in this document was determined in a controlled environment, and therefore, the results that may be obtained in other operating environments may vary significantly. Users of this document should verify the applicable data for their specific environment.

This 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.

The following terms are trademarks of the International Business Machines Corporation in the United States and/or other countries:

AIX	AS/400
AT	CT
DB2	IBM
Netfinity	RS/6000
System/390	XT
400	

The following terms are trademarks of other companies:

Tivoli, Manage. Anything. Anywhere.,The Power To Manage., Anything. Anywhere.,TME, NetView, Cross-Site, Tivoli Ready, Tivoli Certified, Planet Tivoli, and Tivoli Enterprise are trademarks or registered trademarks of Tivoli Systems Inc., an IBM company, in the United States, other countries, or both. In Denmark, Tivoli is a trademark licensed from Kjøbenhavns Sommer - Tivoli A/S.

C-bus is a trademark of Corollary, Inc. in the United States and/or other countries.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and/or other countries.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of

Microsoft Corporation in the United States and/or other countries.

PC Direct is a trademark of Ziff Communications Company in the United States and/or other countries and is used by IBM Corporation under license.

ActionMedia, LANDesk, MMX, Pentium and ProShare are trademarks of Intel Corporation in the United States and/or other countries.

UNIX is a registered trademark in the United States and other countries licensed exclusively through The Open Group.

SET and the SET logo are trademarks owned by SET Secure Electronic Transaction LLC.

Other company, product, and service names may be trademarks or service marks of others.

Appendix F. Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

F.1 IBM Redbooks publications

For information on ordering these publications see “How to get IBM Redbooks” on page 419.

- *Understanding IBM RS/6000 Performance and Sizing,* SG24-4810
- *RS/6000 Performance Tools in Focus,* SG24-4989
- *A Practical Guide to Serial Storage Architecture for AIX,* SG24-4599
- *From Multiplatform Operational Data to Data Warehousing and Business Intelligence,* SG24-5174
- D*ata Modeling Techniques for Data Warehousing*, SG24-2238

F.2 IBM Redbooks collections

Redbooks are also available on the following CD-ROMs. Click the CD-ROMs button at http://www.redbooks.ibm.com/ for information about all the CD-ROMs offered, updates and formats.

CD-ROM Title	Collection Kit Number
System/390 Redbooks Collection	SK2T-2177
Networking and Systems Management Redbooks Collection	SK2T-6022
Transaction Processing and Data Management Redbooks Collection	SK2T-8038
Lotus Redbooks Collection	SK2T-8039
Tivoli Redbooks Collection	SK2T-8044
AS/400 Redbooks Collection	SK2T-2849
Netfinity Hardware and Software Redbooks Collection	SK2T-8046
RS/6000 Redbooks Collection (BkMgr)	SK2T-8040
RS/6000 Redbooks Collection (PDF Format)	SK2T-8043
Application Development Redbooks Collection	SK2T-8037
IBM Enterprise Storage and Systems Management Solutions	SK3T-3694

F.3 Other resources

These publications are also relevant as further information sources:

- *IBM DB2 Universal Database Administration Guide: Design and Implementation Version 6,* SC09-2839
- *IBM DB2 Universal Database Administration Guide: Performance Version 6,* SC09-2840
- *IBM DB2 Universal Database Database System Monitor Guide and Reference Version 6,* SC09-2849
- *IBM DB2 Universal Database SQL Reference, Volume 1, Version 6,* SC09-2847
- *IBM DB2 Universal Database SQL Reference, Volume 2, Version 6,* SC09-2848
- *IBM DB2 Universal Database Command Reference Version 6,* SC09-2844
- *A Complete Guide to DB2 Universal Database,* ISBN 1-5586-0482-0
- *Oracle 8 Administrator's Guide*, A58397-01
- *Oracle 8 Performance Tuning Workshop*, 15108
- *Oracle 8 Server Concepts*, A54646-01
- *Oracle 8 Server Tuning*, A54638-01
- *Oracle Performance Tuning - Tips and Techniques*, ISBN 0-0788-2434-6
- *Oracle for AIX Performance Tuning Tips,* A32146-2
- *Oracle 7 Performance Tuning Tips for UNIX,* A22535-2
- *Oracle 8 & UNIX Performance Tuning*, ISBN 0-1390-7676-X
- *Oracle Performance Tuning*, ISBN 1-5659-2048-1
- *Oracle Backup and Recovery Handbook*, ISBN 0-0788-2106-1
- *OCP Training Guide: Oracle DBA*, ISBN 1-5620-5891-6
- *The Relational Model for Database Management*, E.F. Codd
- *Providing OLAP (On-line Analytical Processing) to User-Analysts: AN IT Mandate* (IBM White-Paper, E.F.Codd)
- *AIX Version 4.3 Commands Reference*, SBOF-1877
- *AIX Version 4.3 System Management Guide: Operating System and Devices*, SC23-4126
- *AIX Logical Volume Manager, from A to Z: Introduction and Concepts*, SG24-5432 (to be published at a later date)
- *AIX Logical Volume Manager, from A to Z: Troubleshooting and Commands*, SG24-5433 (to be published at a later date)

F.4 Referenced Web sites

These Web sites are also relevant as further information sources:

- http://www.rs6000.ibm.com/doc_link/en_US/a_doc_lib/aixgen
- http://www.tpc.org
- http://www.software.ibm.com/data
- http://www.db2mag.com
- http://www.idug.org
- http://w3.aixncc.uk.ibm.com
- ftp://ftp.software.ibm.com/ps/products/db2/fixes/english-us/db2aixv61/
- USENET NewsGroup: comp.databases.ibm-db2
- IBM DB2 NewsGroups: news.software.ibm.com
- http://w3.developer.ibm.com/depts/spra/ORACLE/downloads/Oracle_Stor.PDF

How to get IBM Redbooks

This section explains how both customers and IBM employees can find out about IBM Redbooks, redpieces, and CD-ROMs. A form for ordering books and CD-ROMs by fax or e-mail is also provided.

- **Redbooks Web Site** http://www.redbooks.ibm.com/

  Search for, view, download, or order hardcopy/CD-ROM Redbooks from the Redbooks Web site. Also read redpieces and download additional materials (code samples or diskette/CD-ROM images) from this Redbooks site.

  Redpieces are Redbooks in progress; not all Redbooks become redpieces and sometimes just a few chapters will be published this way. The intent is to get the information out much quicker than the formal publishing process allows.

- **E-mail Orders**

  Send orders by e-mail including information from the IBM Redbooks fax order form to:

	e-mail address
In United States	usib6fpl@ibmmail.com
Outside North America	Contact information is in the “How to Order” section at this site: http://www.elink.ibmlink.ibm.com/pbl/pbl

- **Telephone Orders**

United States (toll free)	1-800-879-2755
Canada (toll free)	1-800-IBM-4YOU
Outside North America	Country coordinator phone number is in the “How to Order” section at this site: http://www.elink.ibmlink.ibm.com/pbl/pbl

- **Fax Orders**

United States (toll free)	1-800-445-9269
Canada	1-403-267-4455
Outside North America	Fax phone number is in the “How to Order” section at this site: http://www.elink.ibmlink.ibm.com/pbl/pbl

This information was current at the time of publication, but is continually subject to change. The latest information may be found at the Redbooks Web site.

IBM Intranet for Employees

IBM employees may register for information on workshops, residencies, and Redbooks by accessing the IBM Intranet Web site at http://w3.itso.ibm.com/ and clicking the ITSO Mailing List button. Look in the Materials repository for workshops, presentations, papers, and Web pages developed and written by the ITSO technical professionals; click the Additional Materials button. Employees may access MyNews at http://w3.ibm.com/ for redbook, residency, and workshop announcements.

IBM Redbooks fax 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.

List of abbreviations

ACID	Atomic, Consistent, Independent and Durable
ADSM	ADstar Storage Manager
AIX	Advanced Interactive Executive
ANSI	American National Standards Institute
API	Application Programming Interface
AS/400	Application System/400
ASCII	American National Standard Code for Information Exchange
ATM	Automated Teller Machine
BI	Business Intelligence
BPS	Bufferpool Services
CICS	Customer Information and Control Program
CLP	Command Line Processor
CPU	Central Processing Unit
CWS	Control Workstation
DARI	Database Application Remote Interface
DASD	Direct Access Storage Device
DAT	Digital Audio Tape
DB	Database
DBA	Database Administrator
DBIA	Dynamic Bit-Mapped Indexing Anding
DBM	Database Manager
DDL	Data Definition Language
DLM	Distributed Lock Manager
DML	Data Manipulation Language
DMS	Database Managed Space
DSS	Decision Support System
EE	Enterprise Edition
EEE	Enterprise-Extended Edition
ERP	Enterprise Resource Planning
FCAL	Fiber Channel Arbitrated Loop
FCM	Fast Communication Manager
FDDI	Fiber Distributed Data Interface
GB	Gigabyte
GUI	Graphic User Interface
HACMP	High Availability Clustered Multiple Processing
HAGEO	HACMP for Large Geographies
HPS	High Performance Switch
HW	Hardware
IBM	International Business Machines Corporation
IT	Information Technology
ITSO	International Technical Support Organization
I/O	Input/Output

IPC Inter-Process Communication

ISO International Standards Organization

JBOD Just a Bunch Of Disks

JDBC Java Database Connectivity

JFS Journaled File System

JM Joao Marcos

KB Kilobyte

LAN Local Area Network

LOB Large Object

LPP Licensed Program Product

LRU Least Recently Used

LTG Logical Track Group

LUN Logical Unit Number

LV Logical Volume

LVM Logical Volume Manager

MB Megabyte

MCA Micro Channel Architecture

MPP Massively Parallel Processors

MTS Multi-Threaded Server

MWC Mirrored Write Consistency

NVS Non-Volatile Storage

OEM Oracle Enterprise Manager

OFA Oracle Flexible Architecture

OLAP Online Analytical Processing

OLTP Online Transaction Processing

OPQ Oracle Parallel Query

OPS Oracle Parallel Server

OS Operating System

PC Personal Computer

PCI Peripheral Component Interconnect

PGA Program Global Area

PL/SQL Procedural Language for SQL

PMR Problem Management Record

PP Physical Partition

PSSP Parallel System Support Programs

PTF Program Temporary Fix

PV Physical Volume

RAID Redundant Array of Independent Disks

RDBMS Relational Database Management System

RS/6000 Risc System/6000

RTFM Read The *Flaming* Manual

SA System Administrator

SAT SATisfy itself (SSA token)

SCN System Change Number

SCSI Small Computer System Interface

SGA System Global Area

SID System Identifier

SMIT System Management Interface Tool

SMP Symmetric Multi Processor

SMS System Managed Space

SP	Scalable Parallel
SQL	Structured Query Language
SSA	Serial Storage Architecture
SW	Software
TCP/IP	Transmission Control Protocol/Internet Protocol
TPC-C	Transaction Processing Council - for Online Transaction Processing
TPC-D	Transaction Processing Council - for Decision Support Systems
TPC-H	Transaction Processing Council - for ad-Hoc Queries
TPC-R	Transaction Processing Council - for Reporting Queries
TTY	Teletype Terminal
UDB	Universal Database
UDF	User Defined Functions
UOW	Unit of Work
UP	Uniprocessor
UPS	Uninterruptible Power Supply
VG	Volume Group
VLDB	Very Large Database
VMM	Virtual Memory Manager
VSD	Virtual Shared Disk
VSS	Versatile Storage Server

Index

Numerics

A

B

C

D

E

F

G

H

I

J

K

P

R

S

T

U

V

W

IBM Redbooks evaluation

Database Performance on AIX in DB2 UDB and Oracle Environments
SG24-5511-00

Your feedback is very important to help us maintain the quality of IBM Redbooks. **Please complete this questionnaire and return it using one of the following methods:**

- Use the online evaluation form found at http://www.redbooks.ibm.com/
- Fax this form to: USA International Access Code + 1 914 432 8264
- Send your comments in an Internet note to redbook@us.ibm.com

Which of the following best describes you?
_ Customer _ Business Partner _ Solution Developer _ IBM employee
_ None of the above

Please rate your overall satisfaction with this book using the scale:
(1 = very good, 2 = good, 3 = average, 4 = poor, 5 = very poor)

Overall Satisfaction _____

Please answer the following questions:

Was this redbook published in time for your needs? Yes___ No___

If no, please explain:

_____

_____

_____

_____

What other Redbooks would you like to see published?

_____

_____

_____

Comments/Suggestions: (THANK YOU FOR YOUR FEEDBACK!)

_____

_____

_____

_____

SG24-5511-00
Printed in the U.S.A.

Database Performance on AIX in DB2 UDB and Oracle Environments

SG24-5511-00