FAKULTI SAINS KOMPUTER & TEKNOLOGI MAKLUMAT

# VERIFICATION THE PERFORMANCE OF MICROSOFT SQL SERVER


**Nama** : NAGESWARY GANNAPATHY

**Nombor Matriks** : WET98050


**Kod Kursus** : WXET3182

**Sesi** : 2001/2002

**Penyelia** : MR. TEH YING WAH

**Moderator** : DR. LEE SAI PECK

## ABSTRACT

Microsoft's SQL Server is a client / server-based relational database management system (RDBMS) that uses T-SQL as its dialect of the SQL language. A client / server database is an application that is divided into a part that runs on a server and a part that runs on workstations (clients). The server side provides security, fault-tolerance, performance, concurrency, and reliable backups. The client side provides the user interface.

SQL Server developers have the responsibility for designing and implementing the databases. Designing a good database starts with understanding the client's requirements for the database. SQL Server administrators have the responsibility for the day-to-day tasks of maintaining and managing the databases. SQL Server administration involves backing up databases and restoring them when necessary, setting up and managing users, managing database security, managing the replication environment, tuning the database system, and troubleshooting any problems that arise.

In this proposal, the *Verification on Physical Design of Microsoft SQL Server 7.0 Performance* will be presented. The proposal mostly will touched on the part of tuning and indexes of SQL Server to perform an optimizer performance and shows how it works on retrieving columns and rows of the real datas. For this purpose, a system will be developed to communicate with SQL Server.

# ACKNOWLEDGEMENT

The development of this research regarding about optimization on physical design of Microsoft SQL Server 7.0 has been done through the advice, assistance and contributions of many individuals.

First of all, I would like to express my utmost gratitude to my project supervisor, Mr. Mathew The Ying Wah who has provided me with unlimited support, guidance and advice throughout the whole development stage of this research. I would like to convey special thanks to my friends for helping me to solve some of problems I faced throughout the duration of the project especially Jimmy Tan, Kamaruzzaman, Mrs. Sumathy, Rekha, Kiruben, and Mr. Bikram.

Lastly, sincere thanks to all lecturers and tutors in Faculty of Computer Science and Information Technology, especially Mr. Ang and Mr.Simon for sharing their time and knowledge with me.

# TABLE OF CONTENTS

**CHAPTER 8 : SYSTEM EVALUATION**



**CHAPTER 9 : CONCLUSION**

# LIST OF FIGURES

# LIST OF TABLES

# PROJECT INTRODUCTION

## 1.1 PROJECT OVERVIEW

Most systems administrators don't perform monitoring and optimization functions because they believe they don't have the time. Most of their time is spent on firefighting, that is, troubleshooting problems that have cropped up. It's safe to say that if they had taken the time to monitor and optimize the systems, those problems might never have arisen in the first place. That makes monitoring and optimization proactive troubleshooting, not reactive, as is the norm.

Monitoring allows to find potential problems before the users find them; without it, have no way of knowing how well the system is performing. Performance Monitor can be used to monitor both Windows NT and SQL Server. Some of the more important counters to watch are Physical Disk : Average Disk Queue ( which should be less than 2 ) and SQL Server : Buffer Manager : Buffer Cache Hit Ratio ( which should be as high as possible ).

Query Analyzer allows seeing how a query will affect the system before place it in production. The profiler is used to monitor queries after they have been placed in general use; it is also useful for monitoring security and user activity. Once have used Profiler to log information about query use to a trace

file, can run the Index Tuning Wizard to optimize the indexes.

Once have created all logs and traces, need to archive them. The various log files can be used later for budget justification and trend tracking. One of the primary reasons to do so is to back up requests for additional equipment. One of the most valuable functions of using the archived data for trend tracking is proactive troubleshooting, that is, anticipating – and avoiding – problems before they arise.

SQL Server has the ability to dynamically adjust most of its settings to compensate for problems. It can adjust memory use, threads spawned, and host of other settings. In some cases, unfortunately, those dynamic adjustments may not be enough and may need to make some manual changes.

Performance tuning in the client / server world is something of a magical art. A combination of so many factors can make an application perform well, and knowing where to focus the time is what 's most important.

The most critical part of optimizing performance is good documentation. Document statistically how the system works or performs before even starting any performance tuning. As the performance tuning cycle begins, should monitor and document the effects of all changes so that it's easy to determine which changes were positive and which were negative. Never assume that all

changes made for one application automatically apply to another application. If want the best results from optimizing SQL Server, need to know and use the proper techniques. If don't, the end result will not be what are hoping for – or what are needed.

## 1.2 OBJECTIVES

The *Verfication on Physical Design of Microsoft SQL Server 7.0 Performance* is a research on physical design to optimize the performance of SQL server by using indexes and tuning. It evaluates the costs of the available methods for retrieving the data and uses the most efficient method. The objectives of the system are to :

(a) To find the storage location of the rows needed by uses an index and extracts only the needed rows.

(b) Selecting a physical database design that is appropriate for the system workload.

(c) To improve performance so frequently accessed procedures do not need to be recompiled.

(d) To provide all the intimate details required wringing out every last transaction per second (tps) possible.

(e) To estimate the amount of disk space required by sizing an SQL Server.

(f) To provides complete and up–to–date statistics to help manage and monitor the performance of SQL Server.

## 1.3 PROJECT SCOPE

The *Verification on Physical Design of Microsoft SQL Server 7.0 Performance* is more to a research on algorithm that is used to evaluate the methods for retrieving the data in the most efficient methods. SQL Server has a reasonably constant execution time and has no way of influencing their execution time. Broadly speaking, there are five techniques, which can use to decrease execution time, which is known as optimization. These techniques are :

1) Retrieve every rows of data from a table. Compare time taken to retrieve a single row from a table.
   Note: compare using different size

2) Retrieve every columns of data from a table. Compare time taken to retrieve a single column from a table.
   Note: compare using different size

3) Compare using STORE PROCEDURE to SQL statements if both of them giving the same results.
   Note: compare using different size

4) Compare indexing and without indexing. What's the time variance to retrieve the same data requested.

5) Comparing the time used on different data types. Analyze the effect of not using the proper data type. Like some fields can use byte instead of integer

## 1.4 PROJECT DEVELOPMENT METHODOLOGY

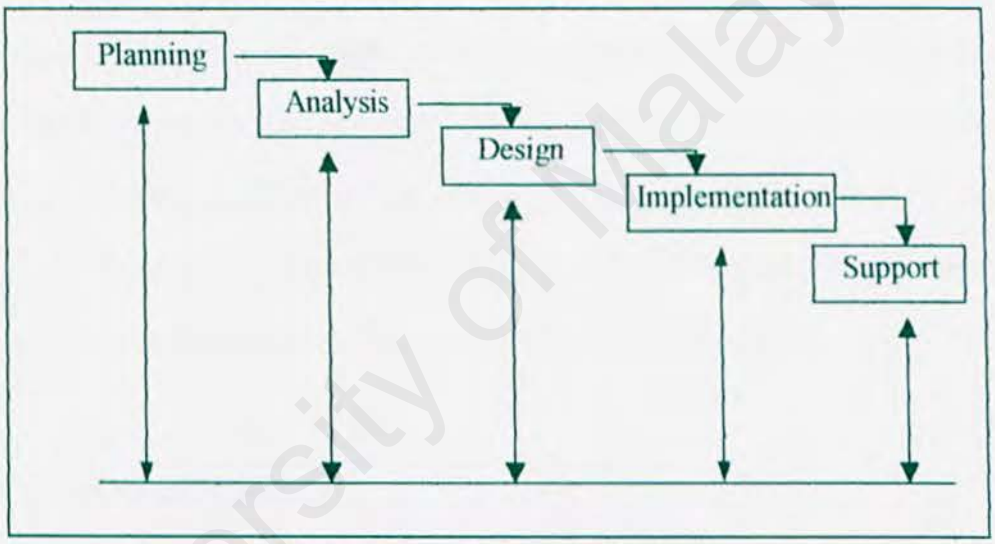



**Figure 1.1 Waterfall Model**

The project development methodology of *Verification on Physical Design of Microsoft SQL Server 7.0 Performance* is Waterfall approach. The Waterfall model builds correction pathways into the model that enable a return to a previous phase. It is the most widely used methodology to implement the system development life cycle ( Meyer, Baber and Pfaffenberger, 1999 ). As

shown in the Figure 1.1, the methodology consists of five phases including planning, analysis, design, implementation and support.

In the planning phase, the current problem will be identified, the need of the project will be recognized and the project objectives will be set. The analysis phase involves the processes of analyzing the existing methods for retrieving data and determining the new most efficient method for extracts the only needed rows. After the system analysis will be the system design. The design phase concerns on the system architecture, database design, as well as the outcome of the reports and screens. The system program design is followed by the system implementation where the system program will be developed and tested for execution. In the final phase, the new system program will be ensured that it has met their goal that is to communicate with SQL Server.

## 1.5 PROJECT SCHEDULE

Project Scheduling plays an important role in planning and developing the entire thesis. It specifies all the activities involve in project development and the duration of time for each activity to successfully implement the project.

(a) *Problem Definition* :

- Recognize the need for the project and the current problem encountered by the most systems administrators to perform monitoring and optimization functions.
- Research on the historical development of SQL Server.
- Develop a project-planning schedule.

(b) *User Requirement Study* :

- Analysis of physical design's capabilities and thinking as well as their search behavior.
- Research on optimization of Microsoft SQL Server requirements.

(c) *Existing Systems Analysis and Documentation* :

- A study of current existing methods to execute data archiving includes all the good functions in the new methods.

(d) *System Requirement Determination* :

- Determine the functional and non-functional requirement as well as the project scope.
- State the innovations that need to occur for the system to be adapted to verify the performance of SQL Server.

(e) *System Analysis* :

- Obtain information through research and observation on the target books.
- Research through the Internet.
- Analysis on the developing tools used by the various systems.

(f) *System Design* :

- Include the prototyping and module design using real database.
- Figure out an overall picture of the new method working process by using algorithms and data dictionaries.
- Specify the system's outcome.

(g) *System Implementation* :

- Development of algorithm.
  - Design the program using the Visual Basic.
  - Translate the entire algorithm into specific program language instructions.
  - Testing and debug the program to eliminate all errors.

- Testing
  - Application testing by individual

- Acceptance testing that involves users to evaluate the system to see whether the system meet their needs and functions correctly.

(h) *Documentation* :

- Recording all information pertinent to the project.

| ID | Task Name | Period | Start | End | Mac | Apr | May | Jun | Jul | Aug | Sept |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | **Problem Definition** | 6 days | 19/3/01 | 25/3 | | | | | | | |
| 2 | **User Requirement Study** | 14 days | 24/3/01 | 7/4 | | | | | | | |
| 3 | **Existing Information Analysis/Documentation** | 30 days | 24/3/01 | 23/4 | | | | | | | |
| 4 | **System Requirement Determination** | 10 days | 26/3/01 | 5/4 | | | | | | | |
| 5 | **System Analysis** | 18 days | 5/4/00 | 23/4 | | | | | | | |
| 6 | **System Design** | 27 days | 24/4/01 | 21/5 | | | | | | | |
| 7 | **System Implementation** | 120 days | 22/5/01 | 19/9 | | | | | | | |
| 8 | **Documentation** | 184 days | 19/3/01 | 19/9 | | | | | | | |

**Figure 1.2 Project Schedule of *Verification the Performance of SQL Server***

## 1.6 SUMMARY OF THE CHAPTERS

Generally there are seven chapters in this project report. Each chapter contains the information of different phrases of the *Verificatin on Physical Design of Microsoft SQL Server 7.0 Performance* development.

(a) *Chapter 1 : Project Introduction*

This chapter contains the overall project overview, current problem definition, the description of the project includes the objectives, the project scope and project's features as well as the project schedule that specifies the activities that must be completed for the project to succeed.

(b) *Chapter 2 : Microsoft SQL Server [Literature Review]*

All the previous studies analysis and research on the project topic will be represented in current chapter. It will focus on the SQL Server as well as their search behaviour in order to design the optimize performance that appropriate to them. It will also contain all the history development of the SQL Server and all the encountered issues.

(c) *Chapter 3 : Optimization Of SQL Server*

All the needs of optimizing and how to tuning SQL Server will take place in this chapter. It will talks about how to measure the success of

tuning and the power to perform better. Then find out what is the backend program that causes poor performances.

(d) *Chapter 4 : Indexes*

All types of indexes are discussed in this chapter. One of the most important decisions regarding the physical implementation of the database is how the indexes will be built. It will focus on selecting indexing method to know how the table will be queried in most situations.

(e) *Chapter 5 : System Analysis And Design*

This chapter includes all the information obtained from the research. The summary and analysis from the observations will be presented. Besides that, all the functional requirements and non-functional requirements of the project will be concerned. The analysis of the system developing tools includes all the hardware and software will be stated in this chapter. A study of the existing algorithm will be done and the description, analysis and documentation of the research will be present in this chapter as well.

(f) *Chapter 6: System Implementation*

This chapter states all the physical design processes that involve a lot of algorithm and new methods. It concerns on the system architecture,

as well as the outcome of the reports. This chapter contains system coding methodologies and testing strategies used. Besides that, the maintenance procedures undertaken will also be included in the chapter.

(g) *Chapter 7 : Stimulation Results*

This chapter discusses the stimulation outputs of the system. Techniques and the results of new methods together with graph analysis will be present in this chapter 7.

(h) *Chapter 8 : System Evaluation and Conclusion*

Chapter 8 discusses the strengths and limitation of the system. All the the problem encountered and the suggestions or comments will also be presented in this chapter. The overall conclusion and the recommendations for future system enhancement will be stated as well.

(i) *Chapters 9 : Conclusion*

Summary of the performance of Microsoft SQL Server 7.0 and the important of SQL Server in the progressive environment will take place in this chapter 9.

# MICROSOFT SQL SERVER

## 2.1 LITERATURE REVIEW

### 2.1.1 What Is The Literature

Although might think of novels and poetry when hear the word *literature*, for a piece of research the meaning is more specific. In terms of a literature review, *the literature* means the works consulted in order to understand and investigate the research problem.

### 2.1.2 Why Write A Review Of The Literature

The literature review is a critical look at the existing research that is significant to the work that is carrying out. Some people think that it is a summary : this is not true. Although need to summarize relevant research, it is also vital that *evaluate* this work, show the *relationships* between different work, and show how it relates to *the* work. In other words, cannot simply give a concise description of, for example, an article : need to select what parts of the research to discuss (e.g. the methodology), show how it relates to the other work (e.g. What other methodologies have been used? How are they similar? How are they

different?) and show how it relates to *the* work (what is its relationship to the methodology?).

## 2.2 INTRODUCTION TO MICROSOFT SQL SERVER

Microsoft SQL Server 7 is an RDBMS (relational database management system). Relational databases are sometimes referred to as self-defining collections of tables. That's because, from end-user's point of view, data appears to be stored in two-dimensional tables similar to spreadsheets. In addition to data tables, RDBMSs all have special tables called catalogs or dictionaries that contain information about particular databases – hence the *self-defining* part of definition.

SQL Server, like most high-end enterprise RDBMSs, is designed to handle multiple users – thousands concurrently, and lots of data – gigabytes (1 billion bytes) or even terabytes (1 trillion bytes) of data. RDBMSs like SQL server are sometimes called OLTP (online transaction processing systems) because they're built to keep track of complex transactions. The RDBMS software is responsible for keeping things in synch and for being able to perform *auto recovery* in case of system failure.

Today, RDBMSs are also used to build data warehouses and data marts for so-called DSS (decision support systems) and BI (business intelligence) applications. These are usually *read-only* (as opposed to *read-write*) databases that contain data that's been consolidated from multiple sources, both for fast access and to minimize the impact of *ad hoc* (unplanned and unscheduled interactive) queries on production OLTP systems.

In the past, organizations and their IT (information technology) staffs have been a lot better about getting data *into* databases than with giving users or customers access to subsets of data they need in order to do their jobs or make decisions. In today's dynamic marketplace, that's changing. IT departments are expected to deliver data over intranets and to better integrate their organizations' diverse data stores with what can be very complex supply chains.

In other words, users take it for granted that RDBMSs will safeguard their data. But they no longer satisfied to have data disappear into what appears to them to be a series of black holes. They expect IT to make it easier for them to get data back *out* of their databases as well.

RDBMSs (Microsoft SQL Server) have moved into the limelight because they can help IT deliver cost-effective solutions in a timely manner.

### 2.2.1 History Of Microsoft SQL Server On Windows NT

Microsoft initially worked with Sybase Corporation on a version of a SQL server system for OS/2. When Microsoft abandoned OS/2 in favor of its new network operating system, Windows NT, it decided to enhance the SQL server engine from Sybase and help modify the code for Windows NT. The resulting product was Microsoft SQL Server 4 for Windows NT, stabilizing at 4.21.

Over time, Microsoft took over more and more responsibility for the development of SQL Server; by version 6, Microsoft was in complete control of the software. Sybase engineers continued developing their database engine to run on Windows NT (Sybase version 10), while Microsoft developers continued enhancing SQL Server 6 (which quickly turned into version 6.5). Sybase continues to develop its product for Windows NT; Microsoft's current version of SQL Server (SQL Server 7) was officially launched in November of 1998.

### 2.2.2 SQL Server 7.0 Enhancements And Philosophies

One major enhancement to SQL Server 7.0 is that the database engine has become largely self-configuring, self-tuning, and self-managing. LazyWrtier and Read-Ahead Manager are self-tuning. Max Async I/O is likely the only sp_configure option that will need to be initially configured when dealing with

servers with larger amounts of storage. This reduction in tuning requirements saves valuable administrative time that can be applied to other tasks. While it is still possible to manually configure and adjust many of the sp_configure options that were available in previous versions of SQL Server, it is recommended that database administrators allow SQL Server to automatically configure and tune all sp_configure options that SQL Server provides defaults for. This allows SQL Server to automatically adjust the configuration of the database server as factors affecting the database server change. (Examples of such factors include RAM and CPU utilization for SQL Server and other applications running off the database server.)

In versions of SQL Server prior to the 7.0 version, *recovery interval* was also used to tune the checkpoint process. SQL Server 7.0 automatically tunes the recovery interval option. The SQL Server 7.0 default of 0 for recovery interval indicates that SQL Server will take responsibility for automatically monitoring and tuning recovery interval. This default setting will maintain recovery times less than one minute for all databases as long as there are no exceptionally long-running transactions present on the system.

SQL Server Log Manager has changed significantly in SQL Server 7.0 from previous versions of SQL Server. SQL Server 7.0 Log Manager manages its own log cache. This separating of the log file management from the data

cache management brings enhanced performance for both components. SQL Server Log Manager is also capable of performing disk I/O in larger byte sizes than before. The larger I/O size combined with the sequential nature of SQL Server logging help to make disk I/O performance very good for the Log Manager. SQL Server 7.0 automatically tunes the performance of SQL Server Log Manager. There is no longer the need to manually tune the sp_configure option logwrite sleep.

## 2.3 A BRIEF REVIEW OF SQL SERVER ARCHITECTURE

In order to discuss verification on indexes and tuning of the SQL Server, it is necessary to briefly discuss and illustrate some key architectural structures and concepts which are strategic to performance considerations. Moreover, understanding of these structures and concepts is vital to the development of optimal SQL Server solutions.

*Verification on Physical Design of Microsoft SQL Server 7.0* is a knowledge-intensive art. The skillful practitioner must have understanding of :

- The RDB system to be used
- The operating system to be used
- The end users and applications which will access the RDB system

- Solid relational database design concepts
- Solid query design concepts
- Suitable hardware platforms and associated components

Optimization is viewed as the planning for efficient operation with respect to user and application performance requirements and logical database design. Optimization takes place as early as the planning and design phases of the SQL Server solution process and accordingly will address SQL Server optimization from a design and development point of view.

### 2.3.1 Database Internals

In order to build an optimal SQL Server solution, one must be able to take advantage of the characteristics of the system. An understanding of the fundamental building blocks of the database is the best place to start. These building blocks consist of the basic internal structures, which by their nature affect how the SQL Server performs. Hence, this section will present information concerning the low level structures associated with a database in general as well as the tables and indexes which by and large are the focus of the optimization and tuning process at this level.

#### 2.3.1.1 What Is a Database

At the highest level, a SQL Server database is, in essence, a storage area for database objects. These objects represent tables, indexes, views, system - and user-defined datatypes, stored procedures, etc. However, these objects describe what resides in the database, not the elements that constitute the database. A SQL Server database is comprised of fractional space of one or more logical devices, which in turn are broken down into disk fragments, and then possibly segments. This represents the external view of the database as described by the data definition language and system tables such as *sysdevices*, *sysdatabases*, *sysusages*, and *syssegments*. There is also an internal view of the database. This internal view consists of low-level data constructs such as devices, disk fragments, allocation units, extents, and pages. It is this internal view, which is of greatest interest, with respect to understanding how to build optimal SQL Server solutions. Hence, a brief summary of each of the internal structures follows. In addition, a discussion relating these internal structures to the higher-level data objects, will serve as the basis for strategies for the optimal design and tuning of database objects and queries.

***Devices***

Database devices store databases and transaction logs. These devices are stored on disk files and as such, are physical storage allocated to the server. A database device must be at least 2 MB or 1024 2K blocks in size. When created, the device is dived into allocation units, each of which is 256 2K pages or 0.5 MB. Thus, a database device will consist of at least 4 allocation units.

Once a database device has been created, a new row is added to the *sysdevices* table in the Master database. This table contains data relevant to the size, page addressing, logical name, and physical location of the device file. Having created a database device of a specific size, it cannot be changed. Thus, care should be given to the determination of device size.

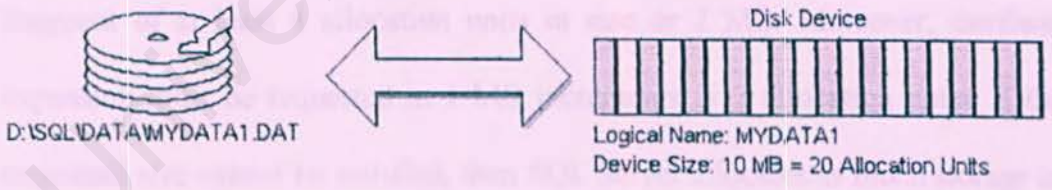



**Figure 2.1 Devices**

***Disk Fragments***

A disk fragment represents the space used by a database over one or more database devices. Each disk fragment of storage for a database must be at least 1 allocation unit in size. However, by default, a new database will require a disk fragment of at least 4 allocation units in size or 2 MB. Moreover, database expansion must be requested in 1 MB increments or 2 allocation units. If the requested size cannot be satisfied, then SQL Server allocates as much storage as possible in 0.5 MB increments or 1 allocation unit. Hence, disk fragment sizes may reflect 0.5 MB increment.

A disk fragment represents the space used by a database over one or more database devices. Each disk fragment of storage for a database must be at least 1 allocation unit in size. However, by default, a new database will require a disk fragment of at least 4 allocation units in size or 2 MB. Moreover, database expansion must be requested in 1 MB increments or 2 allocation units. If the requested size cannot be satisfied, then SQL Server allocates as much storage as possible in 0.5 MB increments or 1 allocation unit. Hence, disk fragment sizes may reflect 0.5 MB increment.

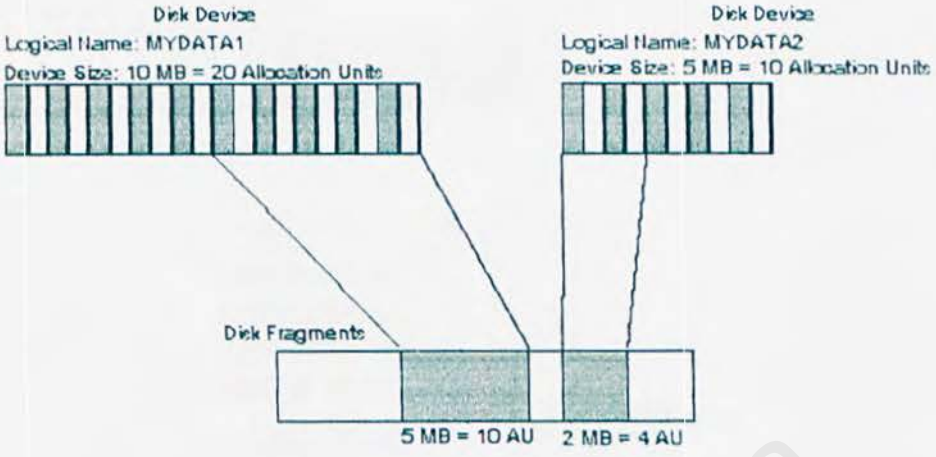



**Figure 2.2 Disk Fragment**

***Allocation Units***

An allocation unit represents 256 contiguous pages or 0.5 MB of internal SQL Server data storage. Within each allocation unit, the first page is the allocation page. It contains an array that shows how the other 255 pages are used. In addition, each allocation unit consists of 32 extent structures.

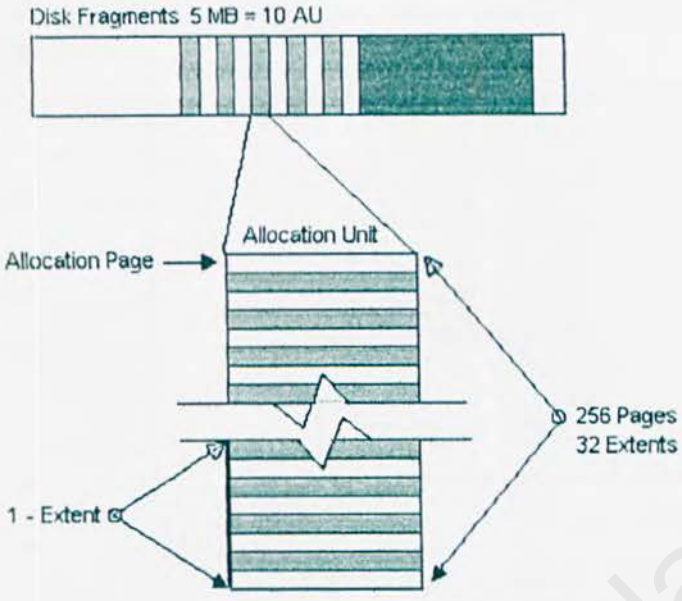



**Figure 2.3 Allocation Unit**

***Extents***

Extents are the smallest unit of data storage reserved for tables and indexes. Each extent consists of eight contiguous pages. 32 extents populate a single allocation unit. Extents are linked together to form a doubly linked circular chain for each table or index object.

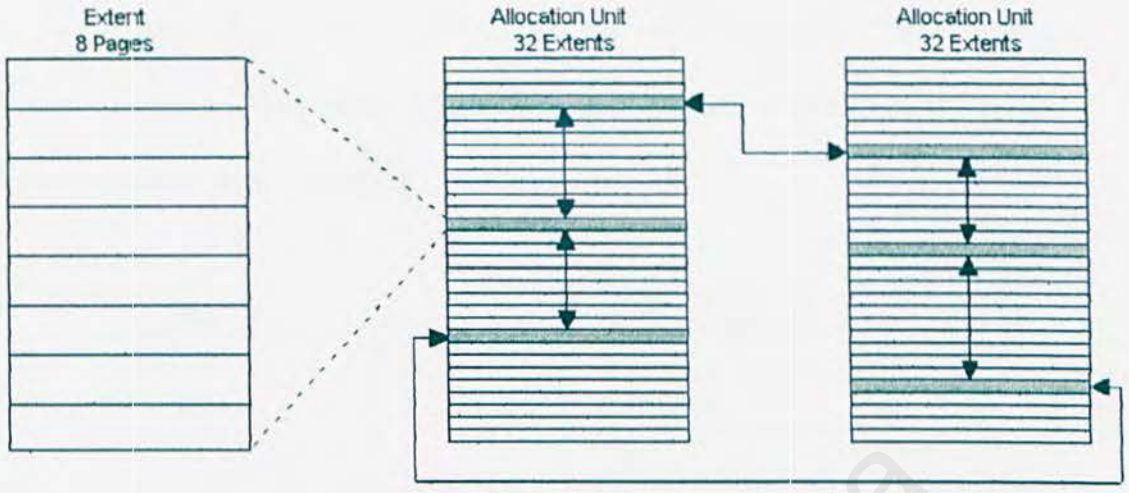



**Figure 2.4 Extent**

***Pages***

SQL Server pages are 2K in size. The page is the unit of I/O for accessing and updating data. There are five distinct types of pages :

- Allocation Pages - Contain information about extents.
- Data Pages - Contain data rows or log records.
- Index Pages - Contain index rows.
- Text/Image Pages - Contain TEXT/IMAGE data.
- Distribution Pages - Contains index key value entries for the purpose of optimizing queries.

Page management is accomplished via extent structures. Thus, from a database table or index object perspective, an allocation of an extent or 8 pages occurs if a new page is required.

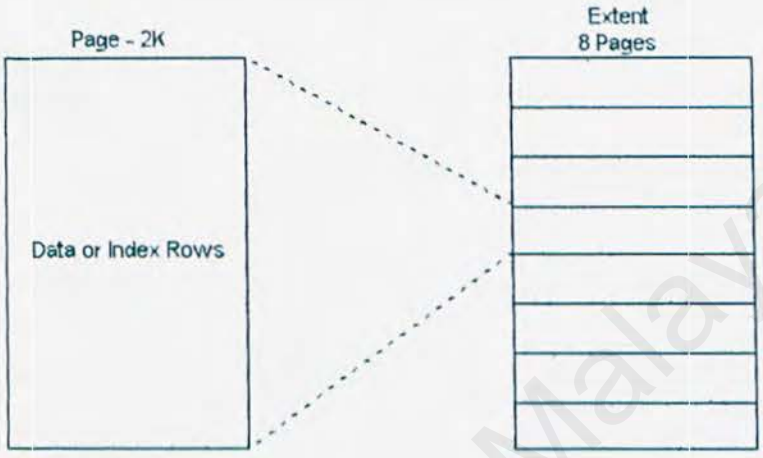



**Figure 2.5 Pages**

***Tables***

SQL Server tables are comprised of data pages. Data pages are 2K in size and are chained together to form a doubly linked list. Each data page contains a 32-byte header, which contains information about the page and links to other pages in the chain. The data rows are stored immediately after the header information and contain row storage information and the actual data. Thus, each data page is capable of storing up to 2016 bytes of data including overhead storage. In addition, a table with no clustered index, will have a row in the *sysindexes* table

which points to the first logical page and the last or root page of the table's data page chain.

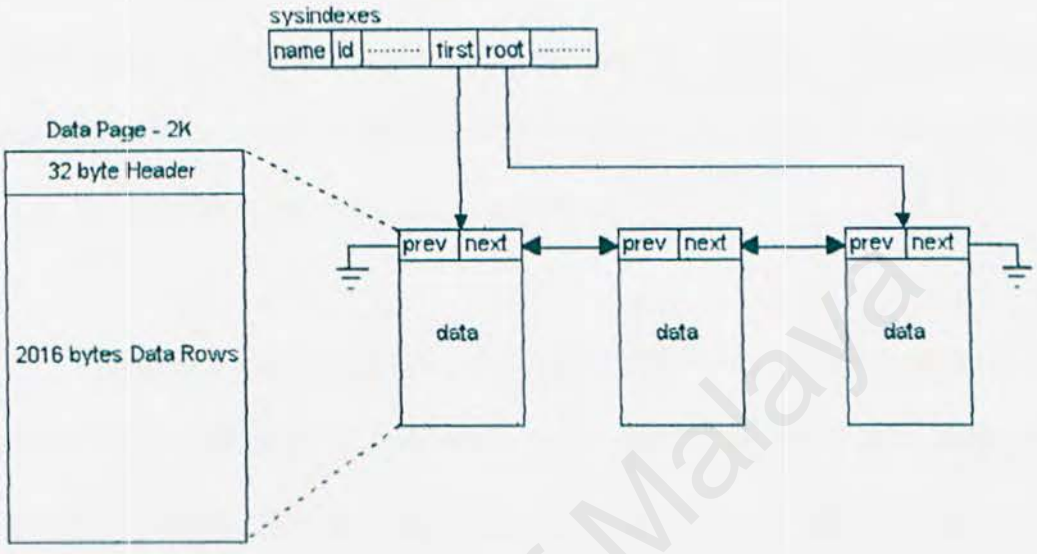



**Figure 2.6 Tables**

All data is stored in a contiguous manner, thereby simplifying data page scans. Information concerning the columns for each row is kept in the *syscolumns* table for each table data object.

***Indexes***

SQL Server indexes are composed of index pages. These pages possess the same physical characteristics as data pages. Index pages consist of a 32-byte header and index rows taking up to 2016 bytes. The index rows contain pointers to index

node pages, data pages, or data rows depending upon the type of index represented. Each page chain in an index is referred to as a level. Index pages at the same level are doubly linked, with the lowest level being the zero (0) level. The highest level is called the root and consists of only one page. These characteristics are true of all SQL Server index types. There are two types of SQL Server indexes, clustered and nonclustered.

Clustered index pages are comprised of index rows containing pointers to other index node pages or data pages at the leaf level. Thus, data pages are physically ordered by the key value associated with a clustered index. In addition, each clustered index will have a row entry in the *sysindexes* table with an *indid* of 1. This entry points to the first data page in the tables chain via its logical page number, and the root page of the clustered index via its logical page number.

Nonclustered index pages are comprised of index rows containing pointers to other index node pages or individual data rows at the leaf level. Hence, data pages are not physically ordered by the key value associated with a nonclustered index. In addition, each nonclustered index will have a row entry in the *sysindexes* table with an *indid* greater than 1. This entry points to the first index page in the leaf level of the index chain via its logical page number, and the root page of the nonclustered index via its logical page number.

### 2.3.2 Network Architecture

The SQL Server communicates with clients via a network interface layer called the Net-Library. This interface layer, along with the Open Data Services layer, account for the bulk of the SQL Server network architecture.

#### 2.3.2.1 Net-Library

A Net-Library is a network interface specific to a particular network interprocess communication (IPC) mechanism. SQL Server uses a common internal interface between Open Data Services, which manages its use of the network, and each Net_Library. By doing this, all network-specific considerations and calls are isolated to just that Net_Library. There may be multiple Net_Libraries loaded, one for each network IPC mechanism in use (for example, one for named pipes, another for TCP/IP sockets, another for SPX, and another for Banyan® VINES® SPP). Unlike SQL Server on other operating systems, Microsoft SQL Server uses this Net_Library abstraction layer at the server as well as the client, making it possible to simultaneously support clients on different networks. Windows NT allows multiple protocol stacks to be in use simultaneously on one system.

It is important to distinguish between the IPC mechanism and the underlying network protocol. IPC mechanisms used by SQL Server include

named pipes, SPX, and Windows® Sockets. Network protocols include NetBEUI, NWLink (SPX/IPX), TCP/IP, and VINES IP. Named pipes, a common IPC used in SQL Server environments, can be used over multiple network protocols (named pipes can be used efficiently over NetBEUI, NWLink SPX/IPX, and TCP/IP, all simultaneously). SQL Server in other environments has traditionally supported only a single IPC mechanism (usually TCP/IP sockets) that was hard-coded directly into the network handler. All clients needed to speak to that IPC mechanism, and nothing else, and usually only with a single network protocol.

#### 2.3.2.2 Open Data Services

ODS functions as the network connection handler for SQL Server for Windows NT. In the past, Open Data Services and the SYBASE® Open Server were often referred to as "conceptually" the front end of SQL Server. However, they were different implementations that attempted to perform the same function. But it is literally true that SQL Server for Windows NT is an ODS application. SQL Server uses the same ODS library (OPENDSNT.DLL) as any other ODS application. ODS manages the network, listening for new connections, cleaning up failed connections, acknowledging "attentions" (cancellations of commands), and returning result sets, messages, and status back to the client. SQL Server clients and the server speak a private protocol known as Tabular Data Stream

(TDS). TDS is a self-describing data stream. In other words, the TDS data stream contains "tokens" that describe column names, datatypes, events (such as cancellation), and return status in the client-server "conversation." Neither clients nor servers write directly to TDS. Instead, open interfaces of DB-Library and ODBC at the client are provided that emit TDS. Both use a client implementation of the Net_Library. At the server side, ODS is basically the mirror image of DB_Library/ODBC.

# VERIFICATION THE PERFORMANCE OF SQL SERVER

## 3.1 VERIFICATION THE PERFORMANCE OF SQL SERVER

There are several ways to measure performance and consequently, there are several parameters to optimize SQL Server. The performance of a database application may be evaluated by one or more of the following criteria :

- Average response time of a transaction
- Maximum response time of a transaction
- Percentage of transactions whose response time does not exceed a certain time limit
- Throughput - the number of transactions per unit of time
- Concurrency - the number of users served simultaneously within a specified response time

The industry-standard TPC benchmarks (issued by the Transaction Performance Council) measure the performance of a system by the number of transactions per second and the average price per transaction. Different TPC benchmarks are designed for different types of applications. TPC-C represents a transaction mix typical for an online transaction processing (OLTP) system, while TPC-D is representative of a data warehouse activity. Without diminishing

the importance of TPC benchmarks, also recommend to perform own testing for the applications on hardware in the environment. TPC results are achieved on specific hardware and network configurations - not to mention that every application and database schema is different and may require specific optimization techniques.

A user's perception of application performance may be the most important factor of its evaluation. Very often, it has little to do with the point of view and may be based on prior experience with legacy systems, comparison with other similar applications and factors unrelated to the database part of the application. For example, an inefficient network may kill the efforts to optimize SQL Server code, a poorly designed GUI may require that a user makes dozens of clicks to receive desired results, and an untrained user may submit an ad hoc query that brings the whole server to a halt. Performance optimization and tuning are related areas of database design. But should keep in mind all of the many components that may influence overall application performance :

- Server hardware
- Server operating system
- Network hardware and topology
- Network operating system
- Application architecture

- Middleware tier
- Client computer hardware
- Frontend applications
- Database design
- Backend (Transact-SQL) programs

From the user's perspective, all these factors contribute to an application's response time. The first task is to find and isolate bottlenecks. If the poor performance is caused by SQL Server, then have some work to do.

It is important to realize that in many cases there is no clear-cut tuning solution. Verification is usually a series of trade-offs. For example, it may sacrifice overall throughput to improve concurrency, or it may optimize average response time at the expense of throughput. In some cases, it might achieve stellar performance of a decision-support query but pay the price by slowing down online transactions. Most importantly, it often have to choose between better performance and lower price. More expensive systems naturally have more power to perform better. As with any other optimization task, tuning SQL Server programs requires that to develop criteria to measure the success of the tuning efforts. It depends on the particular application goals and requirements. Talk to the users to determine what is most important to them. Choose which parameters should not exceed certain limits and which ones are critical to overall

performance. It may even need to assign weights to different criteria reflecting their relative importance.

Sometimes a user's perception of poor response time may be alleviated with an approach that has nothing to do with programming. It could take up to five minutes to bring back results of certain queries because of the large amount of data that had to be processed. There was no more room for SQL code optimization and the client didn't want to pay for faster server hardware. But the angry users turned into happy campers when modified the GUI application to randomly display prestored graphic images while waiting for a query to come back. It allowed users to bring their own favorite pictures and scanned them into a stack of images. The concept was the same as in some modern screensavers, although very novel at the time this application was developed. Users were happy watching pictures of landscapes, animals, and Clint Eastwood (the department manager brought this one himself). Of course, it might argue that times have changed and today they would demand 3-D graphics, animation, sound, and real-time stock quotes.

Any optimization effort requires an investment of time. The day will not stretch beyond 24 hours, no matter how little sleep allow itself. Before spend the precious time on SQL Server code optimization, must find out whether it is the backend program that causes poor performance.

If narrow down the problems to SQL Server, investigate which stored procedure or table is causing the trouble. In many cases, it will find that only one place in the code requires extra work. Should not spend time optimizing pieces that are not causing complaints. Isolating and prioritizing bottlenecks is probably more important than rewriting inefficient queries.

An often-overlooked approach to optimization involves evaluating the workload on the database system and trying to balance it more evenly throughout the day, week and month. Monitor system usage for several weeks to determine the busiest hours of the day and the busiest days of the week and month. Shift all maintenance jobs and long-running reports to off-peak hours and days. It usually requires no code changes but yields incredible performance gains. Empowered with the research results, managers may even consider shifting the working hours of the staff. Spreading work evenly helps to alleviate users' competition for limited server resources during peak hours.

## 3.2 HARDWARE PLANNING FOR PERFORMANCE

Understanding the internal storage structures associated with SQL Server databases, possessing knowledge of the user and application performance requirements, and having designed an optimal logical database design, it is now appropriate to consider the optimal hardware platform based upon this

information. It is the goal of this section to provide information which will help to determine the best possible hardware configuration for the database environment.

### 3.2.1 Hardware Planning

Hardware planning as it relates to SQL Server is primarily concerned with system processors, memory, disk subsystem and the network. These four areas comprise the bulk of all relevant hardware platforms on which Windows NT and SQL Server operate. Hence, this will address planning considerations which are generic to all platforms and useful for scaling and implementing optimal SQL Server solutions.

#### 3.2.1.1 System Processor : CPU

In trying to determine the initial CPU architecture which is right for the particular needs, attempting to estimate the level of CPU bound work which will be occurring on the hardware platform. As far as SQL Server is concerned, CPU bound work can occur when a large cache is available and is being optimally used, or when a small cache is available with a great deal of disk I/O activity aside from that generated by transaction log writes. The type of questions which must be answered at this point are as follows :

- Will the system be dedicated to SQL Server?
- How many users or processes will access the SQL Server?
- What will the level of transaction throughput be?
- Is the SQL Server a departmental system or an enterprise system?
- Will there be a large number of concurrent users accessing the SQL Server?
- Will there be a great deal of aggregation occurring on the SQL Server?

The answer to these questions may have already come from the requirements specifications. If not, it should be able to make some reasonable estimates. The bottom line is purchase the most powerful CPU architecture which can justify. This justification should be based upon the estimates, user requirements, and the logical database design. However, based upon experience it is suggested that the minimum CPU configuration consist of at least a single 80486/50 processor.

### 3.2.1.2 Memory : RAM

Determining the optimal memory configuration for a SQL Server solution is crucial to achieving stellar performance. SQL Server uses memory for its procedure cache, data and index page caching, static server overhead, and configurable overhead. SQL Server can use up to 2 GB of virtual memory, this

being the maximum configurable value. In addition, it should not be forgotten that Windows NT and all of its associated services also require memory.

Windows NT provides each Win32® application with a virtual address space of 4 GB. This address space is mapped by the Windows NT Virtual Memory Manager (VMM) to physical memory, and can be 4 GB in size dependent upon the hardware platform. The SQL Server application only knows about virtual addresses, and thus can not access physical memory directly. This is controlled by the VMM. In addition, Windows NT allows for the creation of virtual address space which exceeds the available physical memory. Therefore, it is possible to adversely affect performance of SQL Server by allocating more virtual memory than there is available physical memory. Hence, the following table contains rule-of-thumb recommendations for different SQL Server memory configurations based upon available physical memory.

| Machine Memory (MB) | SQL Server Memory (MB) |
|---|---|
| 16 | 4 |
| 24 | 6 |
| 32 | 16 |
| 48 | 28 |
| 64 | 40 |
| 128 | 100 |
| 256 | 216 |
| 512 | 464 |

**Table 3.1 Memory Configurations**

These memory configurations are made for dedicated SQL Server systems, and should be appropriately adjusted if other activities such as file and print sharing, or application services will be running on the same platform as SQL Server. However, in most cases it is recommended that a minimum physical memory configuration of 32 MB be installed. Such a configuration will reserve at least 16 MB for Windows NT. Again, these memory configuration recommendations are only guidelines for initial configuration estimates, and will most likely require appropriate tuning. Nevertheless, it is possible to make a more accurate and optimal estimate for SQL Server memory requirements based upon the previous knowledge gained from user and application performance requirements.

In order to make a more accurate estimate for an optimal memory configuration, the following table for SQL Server for Windows NT configurable and static overhead memory requirements have to refer.

| Resource | Configurable | Default Value | Bytes per resource | Space (MB) |
|---|---|---|---|---|
| User Connections | Yes | 25 | 18,000 | 0.43 |
| Open Databases | Yes | 10 | 650 | 0.01 |
| Open Objects | Yes | 500 | 72 | 0.04 |
| Locks | Yes | 5,000 | 28 | 0.13 |
| Devices | No | 256 | 300 | 0.07 |
| Static server overhead | No | N/A | /2,000,000 | 2.0 |
| **TOTAL Overhead** | | | | **2.68** |

**Table 3.2 Overhead Memory Requirements**

Can use this information to calculate a more exact memory configuration estimate with respect to actual memory usage. This is done by taking the calculated TOTAL Overhead above and applying it to the following formula :

SQL Server Physical Memory - TOTAL Overhead = SQL Server Memory Cache

The SQL Server memory cache is the amount of memory which is dedicated to the procedure cache and the data cache.

The procedure cache is the amount of the SQL Server memory cache which is dedicated to the caching of stored procedures, triggers, views, rules, and defaults. Consequently, if the system will take advantage of these data objects and the stored procedures are to be used by many users, then this value should be proportional to such requirements. Furthermore, these objects are stored in the procedure cache based upon the frequency of their use. Thus, it want the most utilized data objects to be accessed in cache verses retrieval from disk. The system default is 20 percent of the available memory cache.

The data or buffer cache is the amount of the SQL Server memory cache which is dedicated to the caching of data and index pages. These pages are stored to the data cache based upon the frequency of their use. Therefore, the data cache must to be large enough to accommodate the most utilized data and index pages

without having to read them from disk. The system default is 80 percent of the available memory cache.

Accordingly, the following example for a dedicated SQL Server illustrates a more accurate estimate of SQL Server memory requirements.

| Resource | Estimated Value | Bytes per resource | Space (MB) |
|---|---|---|---|
| User Connections | 50 | 18,000 | 0.9 |
| Open Databases | 10 - Default | 650 | 0.01 |
| Open Objects | 500 - Default | 72 | 0.04 |
| Locks | 15,000 | 28 | 0.42 |
| Devices | 256 | 300 | 0.07 |
| Static server overhead | N/A | /2,000,000 | 2.0 |
| **TOTAL Overhead** | | | **3.44** |

**Table 3.3 SQL Server Memory Requirements**

- Physical system memory = 48 MB
- Windows NT physical memory = 16 MB
- SQL Server physical memory = 32 MB
- 32 MB - 3.44 MB = 28.56 MB Total Memory Cache
- Procedure cache: 28.56 * 0.2 = 5.712 MB
- Data cache: 28.56 * 0.8 = 22.848 MB

Hence, as a result of such overhead requirements, it will have approximately 28 MB to work with on the SQL Server. As overhead requirements such as user connections and locks grow, this value will be reduced

and may subsequently lead to performance problems which will then require tuning.

### 3.2.1.3 Disk Subsystem

Achieving optimal disk I/O is the most important aspect of designing an optimal SQL Server solution. The disk subsystem configuration as addressed here consists of at least one disk controller device and one or more hard disk units, as well as consideration for disk configuration and associated file systems. The goal is to select a combination of these components and technologies which complements the performance characteristics of the SQL Server. Hence, disk subsystem I/O as it relates to reads, writes, and caching defines the performance characteristics which are most important to SQL Server. The disk subsystem components and features should look for are as follows :

- Intelligent fast SCSI-2 disk controller or disk array controller
  - Controller memory cache
  - Bus Master card - Processor on-board results in fewer interrupts to the system CPU(s).
  - Asynchronous read and write support
  - 32-bit EISA or MCA

- Hardware level RAID supported

- Fast SCSI-2 drives

  - Read ahead caching (at least a track)

The determination of how many drives, of what size, of what configuration, and of what level of fault tolerance, is made by looking back to the user and application performance requirements, understanding the logical database design and the associated data, and understanding the interplay between system memory and disk I/O with respect to Windows NT and SQL Server. There are several key concepts and guidelines which aid in selection of an appropriate disk subsystem components.

*Concept 1* : Most database I/O's (reads and writes) are random with respect to data and indexes. This is true for on-line transaction processing and decision support systems.

*Concept 2* : Writes to the SQL Server transaction log are sequential and occur as large bursts of page level I/O during the checkpoint process or update, insert, or delete operations.

*Concept 3* : Optimal access to randomly accessed data and indexes is achieved by distributing the database over several physical disk units, in a single stripped volume (RAID 0 or RAID 5). This results in multiple heads being able to access the data and indexes.

*Concept 4* : Optimal access to sequentially accessed data is achieved by isolating it from the randomly accessed data and index volume(s), on separate physical disk units which may be RAID configured (usually RAID 1, mirrored for logs). Sequential access is faster via a single head which is able to move in one direction.

*Concept 5* : Duplexing of intelligent disk controllers (SCSI or Array) will usually yield greater performance. This is especially true of systems which must sustain high transaction throughputs, systems with small data (buffer) caches, and systems with large data volumes. In addition, if the number of physical disk units exceed a controllers capacity, another controller will be necessary.

*Concept 6* : A good method for determining the number of the disk units required for an optimal disk subsystem is to multiply the number of I/O's per application transaction by the total number of application transactions per second, as generated by the users or the applications. This will yield the total number of user or application generated I/O's per second. Take this value and

divide it by the average sustainable I/O's per second of the physical disk units may use (an average range is between 30 and 50 I/O's per second, which includes system overhead and latency). The result is a recommended number of disk units for this particular SQL Server solution.

*Concept 7* : A method for determining the size of the disk units required for an optimal disk subsystem is to divide the total space required for the database by number of drives determined previously. This will yield a value which must then be adjusted according to the RAID level employed and or the number of controllers employed.

*Concept 8* : The minimum optimal disk subsystem configuration for any SQL Server solution will consist of the SCSI type of controller and at least two SCSI drives. This disk configuration is necessary in order to isolate the SQL Server transaction log(s), placing them on one physical disk and the database devices or file(s) on the other physical disk.

These concepts should be used as guidelines and not as absolutes. Each SQL Server environment is unique, thereby requiring experimentation and tuning appropriate to the conditions and requirements.

#### 3.2.1.4 Network

As with intelligent disk controllers, the goal is to select an intelligent network interface card (NIC) which will not rob CPU or memory resources from the SQL Server system. This network card should meet the following minimum recommendations :

- 32-bit EISA or MCA
- Bus Master card - Processor on-board results in fewer interrupts to the system CPU(s).
- On-board memory cache

### 3.2.2 Optimize A Hardware Design That Complements The SQL Server Solution

When building an optimal SQL Server solution the choice of hardware should be based upon :

- User and application performance requirements
- Knowledge of the Windows NT operating system
- Knowledge of the SQL Server internals and operations
- The logical database design

All to often, SQL Server solutions are made to fit generically configured hardware platforms, thus resulting in poorly performing systems. Consideration needs to be given to proper hardware configuration before the physical implementation of the database. This strategy will result in less tuning and augmentation of the hardware components, thus resulting in better performance from the beginning and saving time and potentially money.

## 3.3 TUNING THE SQL SERVER SOLUTION

Tuning the SQL Server solution should be much less difficult at this point since have applied optimization guidelines throughout the solution development process. However, reality dictates that no implementation is performance perfect. A prior understanding of the Windows NT Performance Monitor is required in order to apply some of the tuning strategies presented.

All monitoring requires the use of the Windows NT Performance Monitor unless otherwise specified. In addition, all guidelines apply to dedicated SQL Server systems. However, these techniques can be applied to non-dedicated SQL Server systems with only slight modification.

### 3.3.1 Hardware Resource Tuning

The tuning of hardware resources typically involves the discovery of "bottlenecks". A bottleneck is the single resource that consumes the most time during a tasks execution. In the case of SQL Server, such resource bottlenecks adversely affect the performance of normal relational database operations. Hence, the following information pertains to the detection of SQL Server resource bottlenecks, and the subsequent adjustment of the resource in order to relieve the demand and increase performance.

#### 3.3.1.1 Processor Tuning

Processor tuning involves the detection of CPU bound operations. Assuming that SQL Server has sufficient memory to run, increasing CPU power is the most effective hardware-related way to improve performance. It can add CPU power in two ways : using a faster CPU or adding additional CPU(s).

- *Faster CPU* : When performance is important, should consider using as fast a CPU as possible. In general, a faster CPU will probably realize a bigger performance gain over adding an additional CPU. This is because, while adding CPUs provides additional power, the operating system and SQL Server (or any application) incur an

overhead in managing the work performed by multiple CPUs. Of course, once are running the fastest CPU available in chosen architecture, it can add additional CPUs to increase performance.

- *Additional CPU(s)* : Windows NT supports symmetric multi-processing (SMP). Since SQL Server is implemented using native Windows NT threads, it can automatically take advantage of multiple CPUs. SQL Server takes good advantage of SMP platforms, and can boost performance significantly by moving the application to an SMP platform, or adding additional CPUs to an existing SMP platform.

#### 3.3.1.2 Disk Subsystem Tuning

Disk subsystem tuning involves the detection of disk I/O constrained operations. Such bottleneck constraints may be caused by the disk controller, the physical disk drives, or lack of some other resource, which results in excessive disk I/O generating activity. Furthermore, poor disk subsystem performance may also be caused by poor index or database design. The goal is to operate the SQL Server with as few physical I/O's and associated interrupts as possible. In order to monitor low-level disk activity with respect to the **PhysicalDisk** Performance Monitor counters, it is necessary to enable the diskperf option. This can be accomplished by issuing the following command from the system prompt:

*diskperf -y*. Running with this option enabled may result in a slight (0.1%-1.5%) degradation in performance. Hence, disable it when not required for use (*diskperf -n)*.

When performance tuning the SQL Server disk subsystem, first should attempt to isolate the disk I/O bottleneck with the **SQLServer** counters, using the **PhysicalDisk** and **LogicalDisk** counters for more detailed monitoring and refinement of an action plan.

### 3.3.1.3 Networking Tuning

Network tuning with respect to SQL Server performance is affected by the following :

- Throughput of the LAN or WAN.
- Throughput of the server's network interface card.
- Availability of resources on the server to service client requests.

However, when considering remote procedure calls between SQL Servers or data replication, LAN and or WAN throughput will be an important concern.

### 3.3.2 SQL Server Tuning

Tuning the SQL Server involves appropriately adjusting the SQL Server configuration, options and setup values based on observed operational characteristics. Typically these observations are made during peak work cycles in order to optimize for the heaviest workloads. However, application of these recommendations may result in different outcomes depending upon your particular SQL Server environment.

#### 3.3.2.1 Memory

SQL Server memory is divided between SQL Server overhead, the procedure cache and the data cache. The primary goal is to cover SQL Server overhead while effectively distributing the remaining memory between the procedure and data cache via the *procedure cache* configuration parameter. The distribution of the remaining memory between these caches is an exercise in making sure the most utilized objects are cached in their respective caches. Hence, the most utilized stored procedures should be in the procedure cache, while the majority of frequently used indexes and tables should be in the data cache.

The best way to determine how SQL Server is using memory is to execute *DBCC MEMUSAGE*. This command will indicate the amount of memory allocated to SQL Server at startup, the 12 largest objects in the procedure cache, and the 20 largest objects in the data cache. Hence, the following recommendations are based upon the utilization of this data and will aid in determining the optimal size for these caches.

***Tuning the Procedure Cache***

When tuning the procedure cache, the goal is to determine the optimal size for the purpose of holding the most active stored procedures as well as the other procedure cache data objects. In essence, it want to prevent reading stored procedures from the disk as this is very costly. Moreover, if the procedure cache is large enough, it will prevent the displacement of procedures in the cache by procedures not yet in the cache. (Must remember that the SQL Server will store a duplicate copy of each stored procedure execution plan, which is accessed by more than one user.) By default SQL Server distributes 20% of available memory to the procedure cache after SQL Server overhead has been allocated. The task is to determine if this 20% is sufficient, not enough, or to much based on the size of procedure cache objects.

It can determine if the procedure cache is large enough by executing the most frequently used stored procedures, and then running the DBCC MEMUSAGE command, the 12 largest stored procedures in the procedure cache will be displayed. If have more than 12 stored procedures, it can continue to execute the other procedures, checking each time with the DBCC MEMUSAGE command to see if one of the previously executed procedures has remained cached. It can also execute each stored procedure obtaining its execution plan size via the DBCC MEMUSAGE command. Once have executed all high frequency procedures and obtained their sizes, add these size values to derive the total cache size necessary for all procedures.

***Tuning the Data Cache***

The data cache is comprised of the memory left after SQL Server overhead and the procedure cache memory requirements have been satisfied. The goal is to have enough cache space to hold the majority of all indexes used, and a respectable percentage of the most frequently accessed tables, thus reducing physical I/O's.

It also use the DBCC MEMUSAGE command to view the 20 largest objects in the data cache. Again, can use this data in order to determine a

respectable size for the data cache, based on the sizes indicated for these 20 database objects. It can also determine the size of the most frequently accessed tables and indexes. Having calculated these sizes, may elect to allocate enough memory to SQL Server in order to contain the entirety of these database objects in the data cache.

#### 3.3.2.2 TempDB in RAM

If the queries being executed against the SQL Server are using temporary workspace for sort operations, GROUP BY, temporary tables, multitable join operations, etc. it will be beneficial to move ***tempdb*** to RAM. However, in order to make this move, must have enough memory available over that which is already required by and allocated to Windows NT Server and SQL Server. Consequently, if the ***tempdb*** is currently 25 MB in size, then the total memory required on for the system is: 16 MB for Windows NTS + total SQL Server memory + 25 MB for ***tempdb***.

### *3.3.2.3* Other SQL Server Tuning Parameters

***Dedicated Multiprocessor Performance***

If the hardware platform possesses multiple processors and the system will be dedicated to SQL Server, consider enabling the **Dedicated Multiprocessor Performance** option. This will increase SQL Server's performance substantially.

***Boost SQL Server Priority***

If have not enabled the **Boost SQL Server Priority** option, doing so will allow SQL Server threads to run in the real-time priority class (16 - 31). Running at this priority level, SQL Server threads will be executed before all other process threads running in the lower variable priority class (1 - 15). Hence, on single processor machines under heavy load and not dedicated to SQL Server, other processes may be starved. However, if the system is dedicated to SQL Server and disk I/O activity tends to be heavy, enabling this option may result in substantial performance gains.

### 3.3.3 Database Tuning

The database tuning processes is based on performance observations gathered during normal SQL Server operations, typically at the time of peak work cycles. The type of performance problem symptoms which indicate possible database tuning is necessary, usually consists of excessive disk I/O and or excessive cache flushes. Both of these symptoms have been addressed earlier with respect to memory and disk I/O tuning. However, assuming that these SQL Server resources have been sufficiently tuned, it is now time to examine the database design and the mapping of the physical database to physical disk devices.

### 3.3.4 Query and Index Tuning

Under most circumstances, if an efficient database design has been implemented on a suitable hardware platform, the SQL Server query optimizer will efficiently optimize most queries without concern for the structure of such queries, thereby resulting in exceptional performance. However, if a particular query is performing poorly and performance monitoring of critical system resources indicates no bottleneck problems, query and or index tuning may be necessary.

### 3.3.4.1 Analyzing the Query

Analyzing a query is a process of elimination. Before begin to dissect the structure of the SQL query, it is prudent to eliminate other contributing factors which may influence SQL query performance. Having eliminated these other factors, it is then appropriate to analyze the structure of the query.

***Views***

SQL queries that access views may seem relatively simple in structure. Nonetheless, the actual view may be very complex in structure. Consequently, the view may in fact need to be analyzed for poor performance before any queries which access it can be analyzed. Therefore, analyze the underlying SQL statement which comprises the view.

***Triggers***

Slow query performance may also be attributed to the fact that a trigger may be defined for a table associated with the query. Accordingly, it may be that the trigger is performing slowly and not the query. However, trigger overhead is usually very low. The time involved in running a trigger is spent mostly in referencing other tables, which may be in the data cache or on the disk. In fact,

the deleted and inserted tables are always in memory since they are logical tables. Hence, it is the location of other tables which may be causing physical page reads from disk, that is the cause of slow performance.

***Stored Procedures***

The optimization of the search clause by the Query Optimizer is based on the set of values that are given when the stored procedure is first executed. The stored procedure then remains in cache and the query is not re-optimized each time it is executed. An assumption is therefore made that the first set of values used with the stored procedure are representative. If this is not the case, it is necessary to force a recompile of the stored procedure. This can be done by executing **WITH RECOMPILE**, restarting SQL Server, or by executing the **sp_recompile** stored procedure.

Since stored procedures are not re-entrant, if two processes execute a stored procedure concurrently, two copies of the stored procedure are compiled, optimized, and stored in cache. Multiple copies of the same stored procedure may have different query plans if parameter values passed to the stored procedure are very different. The copies stored in cache remain there until they are aged out of cache or until they are forced to recompile, either explicitly by executing sp_recompile or restarting SQL Server, or implicitly by dropping an

index or table that is referenced by the stored procedure. (Stored procedures are automatically recompiled when objects or indexes on any of the tables used by the query plan are dropped. They are not recompiled when add indexes or run **UPDATE STATISTICS**.)

***Concurrency***

A query may be performing slowly due to concurrency conflicts with other queries. In order to determine if concurrency problems do exist, observe if the query runs efficiently during some periods of the normal work cycle and slowly during others. If this behavior is exhibited, then should check the locking levels of other active SQL Server processes.

This can accomplish by running the **sp_who** stored procedure in order to determine if the query being analyzed is being blocked by another process. It can then execute the **sp_lock** stored procedure to obtain more detail on the state of process locks with respect to the identified queries. There are also **SQLServer-Locks** performance monitor counters which are useful for determining the types of locks held system wide. The most useful for determining if server concurrency problems exist are **Total Blocking Locks**, **Total Demand Locks**, and any locks which are "Exclusive". If any of these values are high with respect to the number of executing query processes, then a concurrency problem probably exists.

### 3.3.4.2 Helping SQL Server Choose Indexes

The mere existence of indexes is not enough for SQL Server to use them. Must have to specify conditions in the WHERE clause that take advantage of these indexes. If have only partial conditions for a composite key, specify WHERE clause conditions for the leftmost columns of the key.

For example, suppose have a table with an index on columns a, b, c and d. Depending on the WHERE clause conditions, SQL Server may use all or fewer columns of the index, or not use the index at all, as shown in Table 3.4.

| WHERE Clause Conditions | Key Columns That May Be Used |
|---|---|
| WHERE a = @a AND b = @b AND c = @c AND d = @d | a, b, c, d |
| WHERE a = @a AND b = @b AND c = @c | a, b, c |
| WHERE a = @a AND b = @b AND d = @d | a, b |
| WHERE a = @a AND c = @c AND d = @d | a |
| WHERE b = @b AND c = @c AND d = @d | Index cannot be used (unless all columns needed for the query may be found in the index, in which case SQL Server may do an index scan instead of a table scan) |

**Table 3.4 Usage of Composite Key Columns**

### 3.3.5 Principles For Performance Tuning SQL Server

- *Let SQL Server take care of most of the tuning work.*

  SQL Server 7.0 has been dramatically enhanced in order to create a largely auto-configuring and self-tuning database server. Take advantage of SQL Server's auto-tuning settings. This helps SQL Server run at peak performance even as user load and queries change over time.

- *RAM is a limited resource.*

  A major part of any database server environment is the management of random access memory (RAM) buffer cache. Access to data in RAM cache is much faster than access to the same information from disk. But RAM is a limited resource. If database I/O (input/output operations to the physical disk subsystem) can be reduced to the minimal required set of data and index pages, these pages will stay in RAM longer. Too much unneeded data and index information flowing into buffer cache will quickly push out valuable pages. The driving focus of performance tuning is to reduce I/O so that buffer cache is best utilized.

- *Create and maintain good indexes.*

  A key factor in maintaining minimum I/O for all database queries is to ensure that good indexes are created and maintained.

- *Monitor disk I/O subsystem performance.*

  The physical disk subsystem must provide a database server with sufficient I/O processing power in order for the database server to run without disk queuing. Disk queuing results in bad performance. This document describes how to detect disk I/O problems and how to resolve them.

- *Application and Query Tuning.*

  This becomes especially important when a database server will be servicing requests from hundreds or thousands of connections through a given application. Because applications typically determine the SQL queries that will be executed on a database server, it is very important for application developers to understand SQL Server architectural basics and how to take full advantage of SQL Server indexes to minimize I/O.

- *Take advantage of the powerful combination of SQL Server Profiler and Index Tuning Wizard.*

  SQL Server Profiler can be used to monitor and log a SQL Server's workload. This logged workload can then be submitted to SQL Server Index Tuning Wizard so that index changes can be made to help performance if necessary. Regular use of SQL Profiler and Index Tuning Wizard helps SQL Server perform well as the overall query workload changes over time.

- *Take advantage of SQL Server Performance Monitor to detect bottlenecks.*

  SQL Server 7.0 provides a revised set of Performance Monitor objects and counters, which are designed to provide helpful information for monitoring and analyzing the operations of SQL Server. This document describes key Performance Monitor counters to watch.

- *Take advantage of SQL Server Query Analyzer and Graphical ShowPlan.*

  SQL Server 7.0 introduces Graphical ShowPlan, an easy method to analyze problematic SQL queries. Statistics I/O is another important aspect of Query Analyzer that this document will describe.

## 3.4 SERVER PROCESSES

Processes make up the heart of all transactions, so it makes sense to optimize the processes. The spin counter option and time slice option optimizes processes so that no process can bog down system resources by continuously staying in the system.

The *spin counter* option specifies a limit on the attempts a process can make when trying to obtain access to a resource. On uniprocessor systems, a process can only try once. With multiprocessor systems, however, the default value is 10,000, which can be altered.

The *time slice* option is used to set the time limit for a process to be active. If the process exceeds the time limit, the SQL Server kernel assumes the process is stuck and automatically terminates it. The *time slice* value should carefully consider. If it is too low, it will slow down the system. This drop in system performances happens because the processes scheduling themselves in and out of the CPU bog down the system. If set *time slice* too high, some problems can occur. For example, it can cause long response times when one process doesn't schedule itself out of the CPU until after a long time. This makes the other processes wait in order to be able to run.

To make these changes, use the sp_configure system procedure. Since these are advanced options, it can only change the settings when *show advanced options* is set to one.

# INDEXES

## 4.1 WHAT IS AN INDEX

"What in the world is on indexes and index selection doing in database administration?" It shown that it is important to understand indexes and how SQL Server uses indexes to help developers when they are stuck and come to the DBA looking for some words of wisdom and advice.

An *index* is a separate physical database structure created on a table that facilitates faster data retrieval when search on an indexed column. SQL Server also uses indexes to enforce uniqueness on a row or column in a table or to spread out the data on various data pages to help prevent page contention.

## 4.2 STRUCTURE OF SQL SERVER INDEXES

SQL Server maintains indexes with a B-Tree structure. B-Trees are multilevel self-maintaining structures.

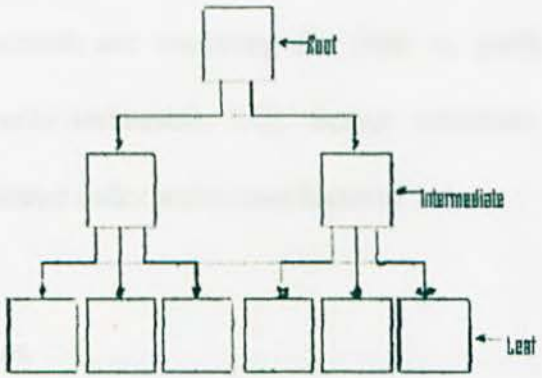



**Figure 4.1 The B-tree Structure**

A B-Tree structure consists of a top level, called the *root*; a bottom level, called the *leaf* (always level 0); and zero to many intermediate levels (the B-Tree in Figure 4.1 has one intermediate level). In SQL Server terms, each square shown in Figure 4.1 represents an index page (or data page).

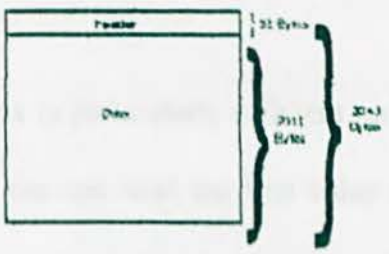


**Figure 4.2 SQL Server Data Page**

The greater the number of levels in the index, the more index pages must read to retrieve the records are searching for (that is, performance degrades as the number of levels increases). SQL Server maintains two different types of indexes: a clustered index and a nonclustered index.

### 4.2.1 Clustered Index

A *clustered index* is a B-Tree structure where level 0, the leaf, contains the actual data pages of the table and the data is physically stored in the logical order of the index. A clustered index determines the storage order of data in a table. A clustered index is analogous to a telephone directory, which arranges data by last name. Because the clustered index dictates the physical storage order of the data in the table, a table can contain only one clustered index. However, the index can comprise multiple columns (a composite index), like the way a telephone directory is organized by last name and first name.

A clustered index is particularly efficient on columns often searched for ranges of values. Once the row with the first value is found using the clustered index, rows with subsequent indexed values are guaranteed to be physically adjacent. For example, if an application frequently executes a query to retrieve records between a range of dates, a clustered index can quickly locate the row containing the beginning date, and then retrieve all adjacent rows in the table

until the last date is reached. This can help increase the performance of this type of query. Also, if there is a column(s) which is used frequently to sort the data retrieved from a table, it can be advantageous to cluster (physically sort) the table on that column(s) to save the cost of a sort each time the column(s) is queried.

Clustered indexes are also efficient for finding a specific row when the indexed value is unique. For example, the fastest way to find a particular employee using the unique employee ID column **emp_id** would be to create a clustered index or PRIMARY KEY constraint on the **emp_id** column.

It is important to define the clustered index key with as few columns as possible. If a large clustered index key is defined, any nonclustered indexes that are defined on the same table will be significantly larger because the nonclustered index entries contain the clustering key.

To understand how the data will be accessed must consider to the using of clustered index for :

- Columns that contain a limited number of distinct values, such as a state column that contains only 50 distinct state codes.

- Queries that return a range of values using operators such as BETWEEN, >, >=, <, and <=.
- Columns that are accessed sequentially.
- Queries that return large result sets.
- Columns that are frequently accessed by queries involving join or GROUP BY clauses; typically these are foreign key columns.
- OLTP-type applications where very fast single row lookup is required, typically by means of the primary key.

Clustered indexes are not a good choice for :

- Columns that undergo frequent changes because this results in the entire row moving (because SQL Server must keep the row's data values in physical order). This is an important consideration in high-volume transaction processing systems where data tends to be volatile.
- Covered queries. The more columns within the search key, the greater the chance for the data in the indexed column to change, resulting in additional I/O.

### 4.2.2 Nonclustered Index

With a nonclustered index, the leaf level pages contain pointers to the data pages and rows, not the actual data (as does the clustered index). A nonclustered index does not reorder the physical data pages of the table. A nonclustered index is analogous to an index in a textbook. The data is stored in one place, the index in another, with pointers to the storage location of the data. The items in the index are stored in the order of the index key values, but the information in the table is stored in a different order (which can be dictated by a clustered index). If no clustered index is created on the table, the rows are not guaranteed to be in any particular order.

Similar to the way use an index in a book, Microsoft® SQL Server™ searches for a data value by searching the nonclustered index to find the location of the data value in the table and then retrieves the data directly from that location. This makes nonclustered indexes the optimal choice for exact match queries because the index will contain entries describing the exact location in the table of the data values being searched for in the queries. If the underlying table is sorted using a clustered index, the location is the clustering key value; otherwise, the location is the row ID (RID) comprised of the file number, page number, and slot number of the row. For example, to search for an employee ID (**emp_id**) in a table that has a nonclustered index on the **emp_id** column, SQL

Server looks through the index to find an entry that lists the exact page and row in the table where the matching **emp_id** can be found, and then goes directly to that page and row.

#### 4.2.2.1 Multiple Nonclustered Indexes

Some books contain multiple indexes. For example, a gardening book can contain one index for the common names of plants and another index for the scientific names because these are the two most common ways in which the readers find information. The same is true for nonclustered indexes. It can define a nonclustered index for each of the columns commonly used to find the data in the table.

To understand how the data will be accessed must consider the using of nonclustered indexes for :

- Columns that contain a high number of distinct values, such as a combination of last name and first name (if a clustered index is used for other columns).
- Queries that do not return large result sets.

- Columns frequently involved in search conditions of a query (WHERE clause) that return exact matches.
- Decision Support System applications for which joins and grouping are frequently required.
- Covered queries.

### 4.2.3 Data Modification And Index Performance Considerations

It is widely known that an index can help speed data retrievals; from time to time, may hear someone say that indexes slow down other operations, such as inserts, updates, and deletes--which is true. It has been mentioned that B-Tree data structures are, for the most part, self-maintaining data structures, meaning that as rows are added, deleted, or updated, the indexes also are updated to reflect the changes. All this updating requires extra I/O to update the index pages.

What happens when a new row is added to a table without a clustered index? The data is added at the end of the last data page. What happens when a new row is added to a clustered index? The data is inserted into the correct physical and logical order in the table and other rows may be moved up or down, depending on where the data is placed, causing additional disk I/O to maintain

the index.As the index pages and data pages grow, they may be required to split, requiring slightly more disk I/O.

In general, should not worry about the time required to maintain indexes during inserts, deletes, and updates. Must be aware that extra time is required to update the indexes during data modification and that performance can become an issue if over-index a table. On tables that are frequently modified, try to restrict the tables to a clustered index and no more than three or four nonclustered indexes. Tables involved in heavy transaction processing should be restricted to from zero to two indexes. If find the need to index beyond these numbers, run some benchmark tests to check for performance degradation.

## 4.3 SUGGESTED INDEX STRATEGIES

Index selection is based on the design of the tables and the queries that are executed against the tables. Before you create indexes, make sure that the indexed columns are part of a query or are being placed on the table for other reasons, such as preventing duplicate data.

### 4.3.1 What To Index

The following list shows criteria that can use to help determine which columns will make good indexes :

- Columns used in table joins
- Columns used in range queries
- Columns used in `order by` queries
- Columns used in `group by` queries
- Columns used in aggregate functions

### 4.3.2 What Not To Index

The following list shows cases in which columns or indexes should not be used or should be used sparingly :

- Tables with a small number of rows
- Columns with poor selectivity (that is, with a wide range of values)
- Columns that are very large in width ( try to limit indexes to columns less than 25 bytes in size)
- Tables with heavy transaction loads (lots of inserts and deletes) but very few decision support operations

- Columns not used in queries

### 4.3.3 Clustered Or Nonclustered Index

As knows, can have only one clustered index per table. Following are some situations in which a clustered index works well :

- Columns used in range queries
- Columns used in `order by` or `group by` queries
- Columns used in table joins
- Queries returning large result sets

Nonclustered indexes work well in the following situations :

- Columns used in aggregate functions
- Foreign keys
- Queries returning small result sets
- When using the DBCC `DBREINDEX` statement to dynamically rebuild indexes (don't have to drop and re-create the index or constraint)
- Information frequently accessed by a specific column in table joins or `order by` or `group by` queries

- Primary keys that are sequential surrogate keys (identity columns, sequence numbers)

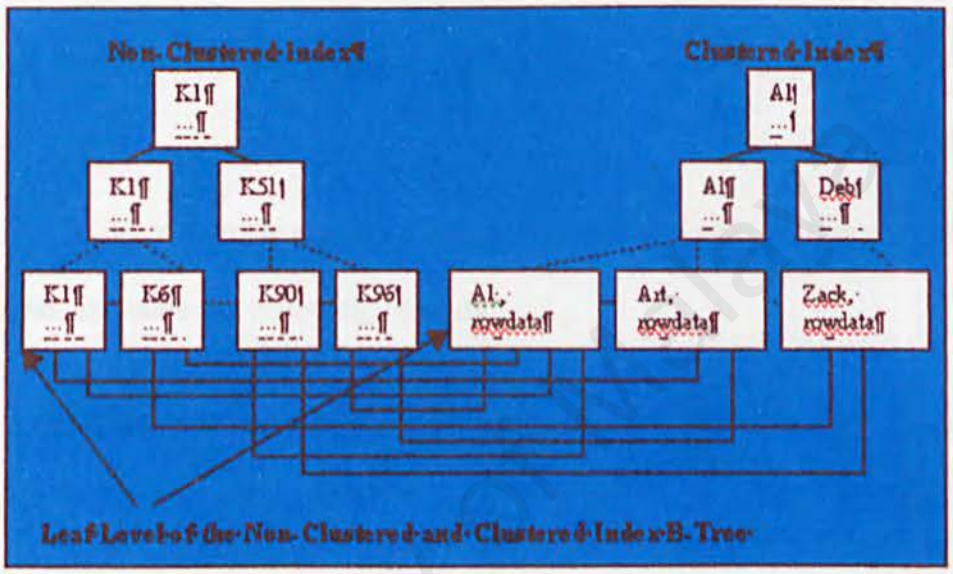



**Figure 4.3 Clustered and Nonclustered Index**

### 4.3.4 Computing Selectivity

It can determine whether a column is a good candidate for an index by doing some simple math and computing the selectivity of the column. First, must determine the total number of rows in the table being indexed. It can obtain this information from the Table Size frame of the Manage Indexes dialog box or by using the following SQL command :

Select COUNT(*) FROM table_name

Then must determine the number of unique values for the column you want to index. To determine this number, execute the following SQL command :

Select COUNT(DISTINCT column1_name) FROM table_name

To determine the number of expected rows returned by using the indexed column, calculate the following formula :

expected number of rows = (1/number of unique values) * Total number of rows in the table

If the expected number of rows is low compared to the total number of rows in the table, the column is a good candidate for an index. This can further validate by computing a percentage, as follows :

Percentage of rows returned = (expected number of rows/total number of rows in the table) * 100

### 4.3.5 Composite Indexes

Composite indexes are indexes created with two or more columns (the maximum number of columns for an index is 16 columns). SQL Server keeps distribution page information on all the columns that make up the composite index, but the histogram of data distribution used by the query optimizer is kept only on the first key, so the order of the keys *does* matter. Use the key with the most unique values as the first key (best selectability). Try not to get carried away by creating composite indexes with a large number of columns. (Try to keep them under four columns.) Too many columns affect performance and make the index key large, increasing the size of the index and requiring to scan more data pages to read the index keys.

### 4.3.6 Index Covering

Index covering is a term used to explain a situation in which all the columns returned by a query and all the columns in the `WHERE` clause are the key columns in a single nonclustered index. SQL Server does not have to read the data pages to satisfy the query; instead, it returns the values on the leaf page of the index. In some cases, a covered nonclustered index can outperform a clustered index.

The downside to index covering is the added overhead to maintain the indexes. Also, it is very difficult to create indexes to cover the many different queries executed by users. Avoid creating indexes to cover queries. It is better off creating single-column or narrow composite indexes for the query optimizer to use.

## 4.4 CREATING USEFUL INDEXES

Creating useful indexes is one of the most important tasks can do to achieve good performance. Indexes can dramatically speed up data retrieval and selection, but they are a drag on data modification because along with changes to the data, the index entries must also be maintained and those changes must be logged. The key to creating useful indexes is understanding the uses of the data, the types and frequencies of queries performed, and how queries can use indexes to help SQL Server find data quickly. A CRUD chart or similar analysis technique can be invaluable in this effort. It might want to quickly review the difference between clustered and nonclustered indexes because the difference is crucial in deciding what kind of index to create.

Clustered and nonclustered indexes are similar at the upper (node) levels—both are organized as B-trees. Index rows above the leaf level contain index key values and pointers to pages the next level down. Each row keeps track

of the first key value on the page it points to. Figure 4.4 shows an abstract view of an index node for an index on a customer's last name. The entry *Johnson* indicates page 1:200 (file 1, page 200), which is at the next level of the index. Since *Johnson* and *Jones* are consecutive entries, all the entries on page 1:200 have values between *Johnson* (inclusive) and *Jones* (exclusive).

| Key | Page Number |
|---|---|
| Jackson | 1:147 |
| Jensen | 1:210 |
| Johnson | 1:200 |
| Jones | 1:186 |
| Juniper | 1:202 |

**Figure 4.4 An Index Node Page.**

The leaf, or bottom, level of the index is where clustered and nonclustered indexes differ. For both kinds of indexes, the leaf level contains every key value in the table on which the index is built, and those keys are in sorted order. In a clustered index, the leaf level is the data level, so of course every key value is present. This means that the data in a table is sorted in order of the clustered index. In a nonclustered index, the leaf level is separate from the data. In addition to the key values, the index rows contain a bookmark indicating where to find the actual data. If the table has a clustered index, the bookmark is the clustered index key that corresponds to the nonclustered key in the row. (If the clustered key is composite, all parts of the key are included.)

In SQL Server clustered indexes are guaranteed to be unique; if don't declare them as unique, SQL Server adds a uniqueifier to every duplicate key to turn the index into a unique composite index. If index on last name is a nonclustered index and the clustered index on the table is the zip code, a leaf-level index page might look something like Figure 4.5. The number in parentheses after the zip code is the uniqueifier and appears only when there are duplicate zip codes.

| Key | Locator |
|---|---|
| Johnson | 98004(1) |
| Johnson | 06801(3) |
| Johnston | 70118 |
| Johnstone | 33801(2) |
| Johnstone | 95014(2) |
| Johnstone | 60013 |
| Jonas | 80863(2) |
| Jonas | 37027 |
| Jonasson | 22033(4) |

**Figure 4.5 A Leaf-Level Index Page.**

### 4.4.1 Tailor Indexes To Critical Transactions

Indexes speed data retrieval at the cost of additional work for data modification. To determine a reasonable number of indexes, must consider the frequency of updates vs. retrievals and the relative importance of the competing types of work.

If system is almost purely a decision-support system (DSS) with little update activity, it makes sense to have as many indexes as will be useful to the queries being issued. A DSS might reasonably have a dozen or more indexes on a single table. If have a predominantly online transaction processing (OLTP) application, need relatively few indexes on a table -probably just a couple carefully chosen ones.

To achieve index coverage in queries must look for opportunities, but don't get carried away. An index "covers" the query if it has all the data values needed as part of the index key. For example, if have a query such as *SELECT emp_name, emp_sex FROM employee WHERE emp_name LIKE 'Sm%'* and have a nonclustered index on *emp_name*, it might make sense to append the *emp_sex* column to the index key as well. Then the index will still be useful for the selection, but it will already have the value for *emp_sex*. The optimizer won't need to read the data page for the row to get the *emp_sex* value; the optimizer is smart enough to simply get the value from the B-tree key. The *emp_sex* column is probably a *char(1)*, so the column doesn't add greatly to the key length, and this is good.

Every nonclustered index is a covering index if all are interested in is the key column of the index. For example, if have a nonclustered index on first name, it covers all these queries :

- Select all the first names that begin with *K*.
- Find the first name that occurs most often.
- Determine whether the table contains the name *Melissa*.

In addition, if the table also has a clustered index, every nonclustered index includes the clustering key. So it can also cover any queries that need the clustered key value in addition to the nonclustered key. For example, if nonclustered index is on the first name and the table has a clustered index on the last name, the following queries can all be satisfied by accessing only leaf pages of the B-tree :

- Select Tibor's last name.
- Determine whether any duplicate first and last name combinations exist.
- Find the most common first name for people with the last name *Wong*.

It can go too far and add all types of fields to the index. The net effect is that the index becomes a virtual copy of the table, just organized differently. Far fewer index entries fit on a page, I/O increases, cache efficiency is reduced, and much more disk space is required. The covered queries technique can improve performance in some cases, but should use it with discretion.

Indexes are also important for data modifications, not just for queries. They can speed data retrieval for selecting rows, and they can speed data retrieval needed to find the rows that must be modified. In fact, if no useful index for such operations exists, the only alternative is for SQL Server to scan the table to look for qualifying rows. Update or delete operations on only one row are common; should do these operations using the primary key (or other UNIQUE constraint index) values to be assured that there is a useful index to that row and no others.

A need to update indexed columns can affect the update strategy chosen. For example, to update a column that is part of the key of the clustered index on a table, must process the update as a delete followed by an insert rather than as an update-in-place. When decide which columns to index, especially which columns to make part of the clustered index, consider the effects the index will have on the update method used.

### 4.4.2 Index Columns Used In Joins

Index columns are frequently used to join tables. When create a PRIMARY KEY or UNIQUE constraint, an index is automatically created. But no index is automatically created for the referencing columns in a FOREIGN KEY constraint. Such columns are frequently used to join tables, so they are almost

always among the most likely ones on which to create an index. If primary key and foreign key columns are not naturally compact, consider creating a surrogate key using an identity column (or a similar technique). As with row length for tables, if can keep the index keys compact, it can fit many more keys on a given page, which results in less physical I/O and better cache efficiency. And if can join tables based on integer values such as an identity, it avoid having to do relatively expensive character-by-character comparisons. Ideally, columns used to join tables are integer columns - fast and compact.

Join density is the average number of rows in one table that match a row in the table it is being joined to. It can also think of density as the average number of duplicates for an index key. A column with a unique index has the lowest possible density (there can be no duplicates) and is therefore extremely selective for the join. If a column being joined has a large number of duplicates, it has a high density and is not very selective for joins.

Joins are frequently processed as nested loops. For example, if while joining the *orders* table with *order_items* the system starts with the *orders* table (the outer table) and then for each qualifying order row, the inner table is searched for corresponding rows. For the most common type of join, an equijoin that looks for equal values in columns of two tables, the optimizer automatically decides which is the inner table and which is the outer table of a join. The table

order that you specify for the join doesn't matter in the equijoin case. However, the order for outer joins must match the semantics of the query, so the resulting order is dependent on the order specified.

### 4.4.3 Create Or Drop Indexes As Needed

If create indexes but find that they aren't used, should drop them. Unused indexes slow data modification without helping retrieval. It can determine whether indexes are used by watching the plans produced via the SHOWPLAN options; this is easy if analyzing a large system with many tables and indexes. There might be thousands of queries that can be run and no way to run and analyze the SHOWPLAN output for all of them. An alternative is to use the Index Tuning Wizard to generate a report of current usage patterns. The wizard is designed to determine which new indexes to build, but can use it simply as a reporting tool to find out what is happening in current system.

Some batch-oriented processes that are query intensive can benefit from certain indexes. Such processes as complex reports or end-of-quarter financial closings often run infrequently. If this is the case, creating and dropping indexes is simple. Consider a strategy of creating certain indexes in advance of the batch processes and then dropping them when those batch processes are done. In this

way, the batch processes benefit from the indexes but do not add overhead to OLTP usage.

## 4.5 Between The Lines

Following are some important points to remember about SQL Server indexes :

- SQL Server maintains indexes with a B-Tree structure.
- In a clustered index, the leaf contains the actual data pages of the table and the data is physically stored in the logical order of the index.
- The SQL Server query optimizer selects at most one index to resolve a query.
- Composite indexes are indexes created with two or more columns.
- Indexes selection must be done carefully.

# CHAPTER 5

# SYSTEM ANALYSIS AND DESIGN

# SYSTEM ANALYSIS AND DESIGN

## 5.1 INTRODUCTION

Systems analysis and design seeks to systematically analyze data input or data flow, processing or transforming data, data storage, and information output within the context of a particular business. Systems analysis and design is used to analyze, design, and implement improvements in the functioning of businesses.

Installation of a system without proper planning will lead to great dissatisfaction and frequently causes the system to fall into disuse. Systems analysis and design lends structure to the analysis and design of information systems, a costly endeavor that might otherwise have been done in a haphazard way. It can be thought of as a series of processes systematically undertaken to improve a business through the use of computerized information systems.

## 5.2 FACT FINDING TECHNIQUES

System analysis starts with data collection. Useful information and recommendations are obtained through carrying some efforts of fact finding that also known as requirement determinations. There are several systematic and

structured fact finding techniques in system analysis, including research, Internet surfing and observation.

### 5.2.1 Research

All the research work is approached from the point of view of this system, which involves reviewing periodicals such as journals, books, conference papers and magazines that contain relevant information. All the periodicals were getting from the University of Malaya's main library, National Library of Malaysia and State Library of Selangor.

### 5.2.2 Internet surfing

Surfing the Internet is indeed a good method of fact finding technique. Existing methods at the web help in giving ideas on the features of the system, data that should keep track in the system database and the implementation of the system. The information about available developing tools also can easily get from vendors' web sites. This helps in evaluating and selecting the most suitable tools for *Verification on Physical Design of Microsoft SQL Server 7.0 Performance*.

### 5.2.3 Observation

Observing the information's needs seeking behavior is one of the important information gathering techniques. Through observing the activities of SQL Server, it is able to seek and gain insight about what problems are actually faced by the database administrators as well as to gain information about information's seeking behavior that is unavailable through any other methods.

## 5.3 REQUIREMENTS SPECIFICATION

The requirements specification follows up on general user's requirements to identify the system's functional requirements as well as the non-functional requirements.

### 5.3.1 Functional Requirements

Functional requirements are statements of services that the system should provide, how the system should react to particular inputs and how system should behave in particular situation. The functional requirement also explicitly state what the system should not do. The table below shows the functional requirements of the *Verification Performance of Microsoft SQL Server*.

| Input Functions | Output Functions | Processing Functions | Storage Functions | Control Functions |
|---|---|---|---|---|
| 1. The system must accept the following inputs :<br><br>• Item data<br>• Partition data<br>• Query Database<br><br>2. All the inputs are either key in by using keyboard<br><br>3.The item data can be imported from server | 1. The system must generate the following outputs :<br><br>• Reports<br>• Search results<br>• Time execute<br>• Needed data<br><br>2. The input can view at the screen, print out, export as another format and save into the diskette. | 1. The system must perform the following processes :<br><br>• Sort lists<br>• Search data<br>• Normalize logical database<br>• Analyze slow performance | 1. The system must maintain the following data :<br><br>• Item data<br>• Partition data<br>• Circulation data<br>• Dewey Decimal Classification Data<br><br>2. All the data can be create, amend and delete. | 1. The system must enforce the following controls :<br><br>• Log-in<br>• Database Backup |

**Table 5.1 Functional Requirements**

### 5.3.2 Non-Functional Requirements

Non-functional requirements are essential definitions of system properties and constraints and the standard must be constraints under which a system must operate.

There are a few issues need to be considered when developing the system, including :

- *Robustness* :

  Robustness refers to the quality that causes a system to be able to handle unexpected error and echo back with proper responses. The System will include effective error handling and validation procedures in order to make it robust. Error messages will be displayed if any unexpected error occurs.

- *Response time* :

  The response time should be within a reasonable internal time where all the desirable information should be available to users at any point in time. User should not be kept waiting for a long time for the results.

- *Reliability* :

  The system should be reliable and shall not cause unnecessary downtime of the assessed environment.

- *Multi-user environment* :

  The system should support not only a user but is able to cope with a large amount of people who access the system concurrently.

## 5.4 DEVELOPING TOOLS ANALYSIS

There are a few criteria that considered during the analysis of the developing tools, including :

1. Enable the development of windows applications that work with real database
2. Enable to create professional-looking installation packages for the application

### 5.4.1 Visual Basic 6.0 (VB 6)

Visual Basic 6.0 (VB 6) is the most productive tool for creating high-performance windows application. The integrated Visual Database Tools and a RAD environment promote productivity by allowing fast application development. VB 6 is able to create applications and both client and server-side components that are optimized for throughout by the world-class Visual C++ 6.0 optimized native-code compiler (Office Software International, 2000).

VB 6 introduces the powerful new standard database access method, ADO (ActiveX Data Objects) to access the library database. All OLE DB drivers such as SQL Server™ 6.5+, Oracle 7.3.3+, Microsoft Access, ODBC, and SNA Server have been into VB. The ADO allowing high-speed access to any ODBC- or OLE DB-compliant database (Thayer, 1999).

Visual Basic 6.0 provides a complete set of tools for integrating databases with any application, kwon as Visual Database Tools. The Visual Database Tools of the Visual Basic simplify operations like database design and the extraction of information from a database. They are not programming tools but tools to help in preparing the application for coding. The Data View Window and the Query Designer are two of the Visual Data Tools. The Data View Window provides a way of maintaining any database connections and the Query Designer is an interactive database query interface that enables almost any type of SQL statement designing (Petroutsos, 2000).

A new productive Data Environment Designer in the Visual Basic 6 is a design time tool used to set up data access for the application. It is a very sophisticated interface that allows table-style query design, easy generation of complex SQL code and a live results preview (Internet.com Corporation, 2000).

The Data Report Designer, which is also known as Data Report Utility is a straightforward tool in VB that streamlines the generation of reports. It allows user to preview, print and export report out of the application (Thayer, 1999). Visual Basic 6 supports mobile computing that enables the development of client/server applications that work with databases. The Visual Basic project is able to bundle into a distributable package, either a compressed CAB file or an executable setup program by using the Packaging and Development Wizard. In

addition to creating a standard installation process, it will also include the file and programs to allow users to uninstall the application if they want to (Thayer, 1999).

### 5.4.2 Consideration Of Database

#### 5.4.2.1 Types of Database

| Type | Advantages | Disadvantages |
|---|---|---|
| File Management Program | • Less complex<br>• Less expensive<br>• Ease to use | • Create flat files that unable to link with other files |
| Database Management System | **Relational database**<br>• Integrate data from multiple files<br>• Ensure data integrity<br>• Reduce data redundancy<br>• Security | **Relational database**<br>• Obtain expensive software<br>• Obtain a large hardware configuration |
| | **Object Oriented database**<br>• Can incorporate sound, video, text and graphics<br>• Well suited for multimedia applications | **Object Oriented database**<br>• Very complex<br>• Costly |

**Table 5.2 Types Of Database**

The file management programs and the database management systems (DBMS) are two widely used database types. All the advantages and disadvantages of both database types are listed in the Table 5.2.

The file management programs enable user to create customized databases and to store and retrieve data from these databases. The file management programs are less complex and thus are less expensive and easier to use than database management systems (Meyer, Baber and Pfaffenberger, 1999). However, the file management systems create flat files where the information in a flat file cannot be linked to data in other files. This will violates the rules of avoiding data redundancy and data integrity. Thus it will not be chosen as the system database.

The relational database management systems (RDBMS) can link data from several files. The relational database and object-oriented databases are two of the widely used type of DBMS. The object-oriented database was not considered, as it is well suited for multiple applications that are not the case in the *Verification the Performance of Microsoft SQL Server.*

The reasons of using relational database as the system database are as following:

1. It can work with many separate files and relate all the data (Meyer, Baber and Pfaffenberger, 1999). These ensure the integrity of data and reduce the data redundancy.
2. Both the logical relationships and query language enable users to retrieve in seconds or minutes (McLeod, 1998).
3. It provides multiple levels of security precautions such as passwords, user directories and encryption (McLeod, 1998).

## 5.5 SYSTEM REQUIREMENTS

There are two categories of system requirements for *Verification the Performance of Microsoft SQL Server*, that is the development environment and the runtime environment.

### 5.5.1 Development Environment

#### 5.5.1.1 Hardware Requirements

The hardware requirements for developing the system are listed as following :

- IBM PC or compatible, with Pentium 144MHz processor or higher
- Minimum 24MB RAM (32MB RAM recommended)

- 700 MB of hard disk space or higher
- VGA or higher-resolution monitor
- Printer and label printer as output devices
- Keyboard and mouse as input devices

#### 5.5.1.2 Software Requirements

The system requirements for developing the system are listed as following :

- Microsoft Visual Basic 6.0
- Windows NT operating system
- Crystal Report 4.0

### 5.5.2 Runtime Environment

#### 5.5.2.1 Hardware Requirements

The hardware requirements for running *Virtual Library* are listed as following :

- IBM PC or compatible, with Pentium 500MHz processor or higher
- Minimum 24MB RAM (32MB RAM recommended)

- 400 MB of hard disk space or higher
- VGA or higher-resolution monitor
- Printer and label printer as output devices
- Keyboard and mouse as input devices

#### 5.5.2.2 Software Requirements

The software requirements for running *Optimizing Microsoft SQL Server* are listed as following :

- Windows NT operating system

## 5.6 PROJECT SPECS

As mentioned earlier, the project specs mostly to evaluate available methods for retrieving data and uses the most efficient new methods to achieve the data. To optimize the SQL Server, a host language will be used in a program. For this purpose, Visual Basic is been chosen to work with SQL Server. The host language is not an existing programming language, but a pseudo programming language. This approach has been chosen in order to not get buried in all sorts of details that have nothing to do with the combination of a host languages and the

SQL language. This way is called as embedded SQL. There are three main advantages resulting from this way :

- Testing of embedded SQL statements can be done interpretively.
- Embedded SQL statements are compiled resulting in a performance improvement with respect to interpretive SQL Server.
- End users principally employ interpretive SQL whereas embedded SQL is intended for experienced programmers. The uniformity of both forms of SQL simplifies communications between these two groups.

### 5.6.1 Methods And New Objects

To most effectively optimize Microsoft SQL Server performance, must identify the areas that will yield the largest performance increases over the widest variety of situations, and focus analysis on these areas. Otherwise, it may expend significant time and effort on that may not yield sizable improvements.

Experience shows that the greatest benefit in SQL Server performance can be gained from the general areas of logical database design, index design, query design, and application design. Conversely, the biggest performance problems are often caused by deficiencies in these same areas. If concerned with performance, then should concentrate on these areas first, because very large

performance improvements can often be achieved with a relatively small time investment.

While other system-level performance issues, such as memory, cache buffers, hardware, and so forth, experience shows that the performance gain from these areas is often incremental. SQL Server manages available hardware resources automatically, for the most part, reducing the need of extensive system-level hand tuning.

Most performance problems cannot be successfully resolved with only a server-side focus. The server is essentially a "puppet" of the client, which controls what queries are sent, and thereby what locks are obtained and released. Although some tuning is possible on the server side, successful resolution of performance problems will usually depend on acknowledging the dominant role the client plays in the problem and analyzing client application behavior.



#### 5.6.1.1 VB Source Code

As will discover, VBSQL is pretty easy to use. First, and most importantly, the SQL text itself is loaded with the cmd.CommandText = Trim$

(txtDataWindow.Text) statement. This and the resulting execute method are all that must have to allow for runtime SQL entry and execution.

As mentioned earlier, it's possible to allow the users to execute pre-prepared statement and stored procedures as well. It also can be used to issue "straight-up" SQL such as SELECT* from tblRequest. The power of this little utility should make for some very interesting applications development. Example of VB source code :

```
Private Sub txtDataWindow_KeyPress (KeyAscii As Integer)
    If KeyAscii = vbKeyreturn Then
        Beep
        Call ProcessDataWindow
    End If
End Sub
```

Before a program can actually run, it must be processed by a number of utility program. The two most important tasks of the precompiler are syntax checking and generating host language statements. All SQL statements are checked for their syntactic accurancy. For example, the precompiler will disallow SELECT statements with no FROM clause, or SELECT statements

whose number of column expressions in the SELECT clause is not equal to the number of variables named in the INTO clause.

A second task of the precompiler is to generate statements from SQL statements. The program listing in the previous section is one of these generated programs. At the same time the precompiler builds a database request module (DBRM). It is sometimes referred to as an access module. A DBRM contains all the information about the SQL statements from the program concerned.

The compiler generates an object module from the precompiled program. The link editor then takes the object module and builds a load module. A load module is a program which is now suitable to be loaded into the computer memory for execution. Compilers and link editors do not form any part of a database management system, but are among a number of stand-alone utility programs.

When the program has been processed by link editor and BIND program, it is ready for execution. If, during the execution of a program, a portion of code generated by the processing strategy is fetched from the SQL catalog and used to access the data.

## 5.7 CONCLUSION

SQL Server is capable of very high performance on large databases. To achieve this performance potential, must use efficient database, index, query, and application design. These areas are the best candidates for obtaining significant performance improvement. Try to make each query as efficient as possible, so that when the application scales up to more users, the collective multi-user load is supportable. Study of the client application behavior, the queries submitted by the application, and experimentation with indexes using the guidelines in this document are strongly encouraged. A methodical approach in analyzing performance problems will often yield significant improvement for relatively little time investment.

# CHAPTER 6

# SYSTEM IMPLEMENTATION

# SYSTEM IMPLEMENTATION AND TESTING

## 6.1 INTRODUCTION

System implementation is the acquisition and integration of the physical and conceptual resources that produce a working system (Meyer, Baber and Pfaffenberger, 1999). It is the physical realization of the database and application designs (Connolly and Begg, 1998). There are two main tasks in the system implementation phase that is system development and system testing.

## 6.2 SYSTEM DEVELOPMENT

### 6.2.1 Database Development

The first step in the system development is to develop the system database based on the logical data model for *Verification on Physical Design of Microsoft SQL Server 7.0 Performance* during the system design phase.

The database development is started by creating an empty database called Payroll. A table is then created by specify all the fields for the table and the field properties. A primary key is allocated for the table in the database. After the table being created, relationships between the fields is established to enforce

referential integrity. The referential integrity is an important constraint on a relationship that ensures consistency between related fields.

### 6.2.2 Application Development

Application development involves code generation that translates all the algorithms into Visual Basic program language instructions. Several programming principles have been employed in writing the program to ensure system consistency, maintainability and readability. All the programming principles are as following :

a) Prefix the name of each object with naming convention to make the object easy to identify and thus enhance program readability. The list of some used controls and their prefix are shown in Table 6.1.

Table 6.1 Part of Used Controls and Its Prefix

| Control | Prefix | Function |
|---|---|---|
| Frame | fra | Acts as a container for other controls |
| Label | lbl | Display start and end time |
| Command | cmd | Display options from which the user can do and go for it |
| PictureBox | pic | Also acts as a container for other controls |
| TextBox | txt | Control for user input |
| Ado Data Control | ado | Supports connecting to a database |
| DataGrid | dgd | A table that is populated automatically with data from a database |

b) Choosing meaningful variable names, procedure names and parameter variable names helps a program to be "self-documenting" without excessive use of comments.

c) All declarations are placed at the beginning of procedure and declarations are separated from the executable statements in that procedure with a blank line to make the declarations stand out and contribute to program readability.

d) When calling a sub procedure, use keyword *Call* and enclose arguments being passed in parentheses to improve program readability

e) Insert comments to document the programs and improve program readability

Figure 6.1 shows the sample of coding for retrieving rows of database from SQL Server. All the programming principles have been followed when writing the code.

**Figure 6.1 Sample Code of Verification the Performance of SQL Server**

```
Private Sub ProcessDataWindow ( )
  Dim cmd As adodb.Command
  Dim rs As adodb.Recordset
  Dim objField As adodb.Field
  Dim lMaxRowsToShow As Long
  Dim l as Long
  Dim strMsg As String
  RetrieveCurrentParms
  setHGOn

'set database connection
  Set cmd = New adodb.Command
  cmd.ActivateConnection = "Driver={SQL Server};" & "Server=" & sServer & ";" &
     "Database=" & sDatabase &";" & "Uid=" & sUserID &";" & "PWD=" & sPassword &";"
  cmd.CommandText = Trim$ (txtDataWindow.Text)
  cmd.CommandTimeout = sTimeout
  cmd.Properties ("Maximum Rows") = sMaximumRows
  Set rs = cmd.Execute
  txtDataResults.Text = ""
  If Not rs.EOF Then
     For Each objField In rs.Fields
        strMsg = strMsg & Trim (objField.Name) & Chr (9)
     Next
     strMsg = strMsg & vbCrLf
     txtDataResults.Text = strMsg & vbCrLf
     If txtShowsRows.Text <> "" Then
        lMaxRowsToShow = CLng (txtShowRows.Text)
     End If
     Do While Not rs.EOF
        l = l + 1
        If l> lMaxRowsToShow Then Exit Do
        strMsg = strMsg & Chr (9)
        For Each objField in rs.Fields
           strMsg = strMsg & objField.Value & Chr (9)
        Next
        rs.MoveNext
        strMsg = strMsg & vbCrLf
     Loop
  End If
  lblRowsReturned.Caption = rs.RecordCount
  txtDataResults.Text = strMsg
  rs.Close
  cmd.ActivateConnection.Close
  setHGOff
  Exit Sub
End Sub
```

### 6.2.3 Database Access

This section discusses the concept of database access used in *Verification on Physical Design of Microsoft SQL Server 7.0 Performance* during the implementation phase. It also includes SQL statement which is required for data manipulation .

#### 6.2.3.1 Connecting To Data Stores

In order to access a data store, the first thing I did is to create connection to the database. I used the ODBC Data Sources or DSNs to connect to the data source. It is only applicable to ODBC Data Sources. They are centralized and the Data Source must be a System Data Source. ODBC Data Sources are created from the ODBC Data Source Administrator, which can be found in the Administrative Tools folder.

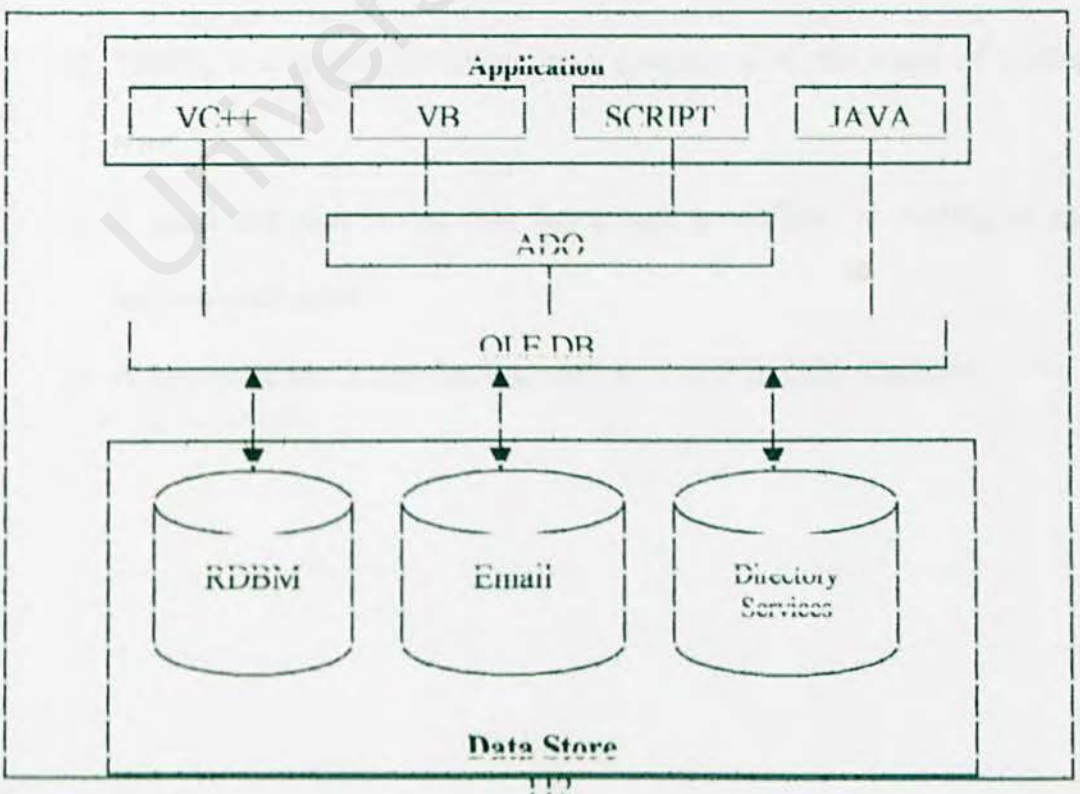
#### 6.2.3.2 Database Access With ADO

ActiveX Data Object (ADO) is a thin layer on top of OLE DB and provides an interface for languages such as Visual Basic and scripting languages, which can't access OLE DB directly. ADO also provides a standard COM interface to OLE DB, which is easily understandable by programmers. Creating data access pages

with ADO yields faster and more efficient system. ADO are the components that allow us to interact with data stores. This might just building a page based on some data or fully interactive e-commerce system.

In database programming, we have ODBC (Open Database Connectivity) and RDO ( Remote Data Objects). ODBC is an Application Programming Language (API) to allow access to relational database such as MS Access and SQL Server. RDO are ActiveX objects that sits on top of ODBC, giving all of the facilities of ODBC, but in an easy to use form. OLE DB is the underlying technology that interfaces between our programs and the source of the data. We can equate OLE DB to ODBC, and ADO to RDO.

**Figure 6.2 OLE DB And ADO Architecture**



#### 6.2.3.3 The ADO Recordset Object

ADO has seven main objects and one of is the *Recordset* object which can be used to store records from a table or the results of executing a query or stored procedure. The *Recordset* objects together with methods such as *movenext* and *movefirst*, makes it can traverse through all the records associated to it.

## 6.3 SYSTEM TESTING

System testing is a verification and validation process. A successful testing will uncover errors in the systems and demonstrates that system functions appear to be working according to specification (Pressman, 2001). Glen Myers (1979) states a number of rules that can serve well as testing objectives :

a) Testing is a process of executing a program with the intent of finding an error

b) A good test case is one that has a high probability of finding an as-yet-undiscovered error

c) A successful test is one that uncovers an as yet undiscovered error

### 6.3.1 Testing Principles

There is a set of testing principles that should understand to guide the system testing. Several testing principles suggested by Davis (1995) have been followed in testing the *Optimization on Physical Design of Microsoft SQL Server* including :

a) All tests should be traceable to the requirement's needs.

b) Test should be planned long before testing began. Testing planning can begin as soon as the requirement model is complete.

c) Testing should begin "in the small" and progress toward testing "in the large". The first test planned and executed generally focus on individual components. As testing progress, focus shifts in an attempt to find errors in integrated clusters of components and ultimately in the entire system.

### 6.3.2 Testing Strategies

There are various testing strategies available to assess completeness and correctness of a system. The newly developed experiment system of *Verification*

*on Physical Design of Microsoft SQL Server 7.0 Performance* is tested thoroughly using different testing strategies that involves a series of test strategies including unit testing, integration testing and error handling debugging.

### 6.3.2.1 Unit testing

Unit testing focuses verification effort on the smallest unit of system design that is system module. During this step, all important control paths are tested to uncover errors within the boundary of the module by using the component level design description as a guide (Pressman, 2001).

**Figure 6.3 Unit Test**


Generally, the unit tests involves five test cases as shown in figure 6.3 and the purpose of each test case is stated below (Pressman, 2001) :

a) Module interface is tested to ensure that information properly flows into and out of the program unit under test.

b) Local data structure is examined to ensure that data store temporarily maintains its integrity during all steps in an algorithm's execution and the local impact on global data should be ascertained during unit testing

c) Boundary conditions are tested to ensure that the module operates properly at boundaries established to limit or restrict processing

d) All the independent paths through the program structure are excised to ensure that all statements in a module have been executed at least once

e) All error handling paths are tested to ensure its ability to detect and recover all fatal errors during system execution

#### 6.3.2.2 Integration testing

Integration testing is a systematic technique for constructing the program structure while at the same time conducting tests to uncover errors associated with the interfacing. Its objective is to take unit tested components and build a program structure that has been dictated by design (Pressman, 2001).

The integration testing was conducted incrementally where the system is constructed and tested in small increments to isolate and correct the errors easier and the interfaces are more likely to be tested completely (Pressman, 2001).

There are a number of different incremental integration strategies available including top-down integration, bottom-up integration, regression testing and smoke testing. Based on the system characteristics and project schedule, a combined approach that uses top down tests for upper levels of the program structure, coupled with bottom-up tests for subordinate levels was selected as *Verification on Physical Design of Microsoft SQL Server 7.0 Performance* system integration testing.

Top-down integration beginning with the main control module as a test driver and stubs are substituted for all components where modules are integrated moving downward through the control hierarchy. Tests are conducted as each component is integrated. Top down integration enables the detection of design error early in the testing phase and avoiding extensive redesign or re-implementation (Connolly and Begg, 1998).

Bottom up integration begins construction and testing with atomic modules where low-level components are combined into cluster to perform a specific system subfunction and tested. Bottom-down integration is an easier test

case design because processing required for component subordinate to a given level is always available and thus the need for stubs is eliminated.

#### 6.3.2.3 Error handling and debugging

Error handling enables the development of clearer, more robust and more fault-tolerant programs. Error handling provides the ability to attempt to recover from infrequent fatal errors rather than letting them occur and suffering the consequences (Deitel, 1999). In the experiment system of *Verification on Physical Design of Microsoft SQL Server 7.0 Performance*, error handling codes only applied at place where errors are likely to occur because it will make the code more difficult to understand and maintain.

Debugging is the process of finding and correcting errors or bugs in the source code of computer program (Meyer, Baber and Pfaffenberger, 1999). There are a number of debugging tools being used in performing the system debugging, including Toggle Breakpoint, Step Into, Add Watch and so on. When debugging the system, the Locals window and Immediate window are used to check the value of variables.

### 6.3.3 Summary

Systems implementation is quite a challenging phase as it includes programming or coding. As I am new to this VB language, I have to read up all the materials for the beginners. I have to try out examples to see how the actual functions of this source language. There are several ways for us to establish connection to the system database. I tried the ODBC Connection because it's quite fine compare to the others which didn't work well for me. Unit testing has been done as a part of implementation phase.

Testing is an important phase in developing systems, because it has dual functions. The testing phase enabled me to defect the errors occurred in my system. Without testing, there is no reason why our system won't work as intended. Besides that, testing also helped me to judge whether or not the program is usable in practice. I came to know that testing only demonstrates the presence of errors and it cannot show that there are no errors in a program.

# STIMULATION RESULTS

## 7.1 STIMULATION

To evaluate the performance of the proposed *Verification on Physical Design of Microsoft SQL Server 7.0 Performance*, simulations are carried out using the Visual Basic 6.0. Several components and functions are developed and incorporated into the system. These experiment shows that the greatest benefit in SQL Server performance can be gained from the general areas of logical database design, index design, query design, and application design. For the purpose of comparison time consuming, three main modules are included in the system that is between columns, between rows and between rows and columns.

## 7.2 USE EFFICIENT INDEX DESIGN

Unlike many non-relational systems, relational indexes are not considered part of the logical database design. Indexes can be dropped, added, and changed without affecting the database schema or application design in any way other than performance. Efficient index design is paramount in achieving good SQL Server performance. For these reasons, should not hesitate to experiment with different indexes.

The optimizer reliably chooses the most effective index in the majority of cases. The overall index design strategy should be to provide a good selection of indexes to the optimizer, and trust it to make the right decision. This reduces analysis time and gives good performance over a wide variety of situations.

The following are index design recommendations :

- **Examine the WHERE clause of SQL queries, because this is the primary focus of the optimizer.**

  Each column listed in the WHERE clause is a possible candidate for an index. If have too many queries to examine, pick a representative set, or just the slow ones. If development tool transparently generates SQL code, this is more difficult. Many of these tools allow the logging of the generated SQL syntax to a file or screen for debugging purposes. Can find out from the tool's vendor if such a feature is available.

- **Use narrow indexes.**

  Narrow indexes are often more effective than multicolumn, compound indexes. Narrow indexes have more rows per page, and fewer index levels, boosting performance.

The optimizer can rapidly and effectively analyze hundreds, or even thousands, of index and join possibilities. Having a greater number of narrow indexes provides the optimizer with more possibilities to choose from, which usually helps performance. Having fewer wide, multicolumn indexes provides the optimizer with fewer possibilities to choose from, which may hurt performance.

It is often best not to adopt a strategy of emphasizing a fully covered query. It is true that if all columns in SELECT clause are covered by a non-clustered index, the optimizer can recognize this and provide very good performance. However, this often results in excessively wide indexes and relies too much on the possibility that the optimizer will use this strategy. Usually, should use more numerous narrow indexes which often provide better performance over a wider range of queries.

Should not have more indexes than are necessary to achieve adequate read performance because of the overhead involved in updating those indexes. However, even most update-oriented operations require far more reading than writing. Therefore, do

not hesitate to try a new index if think it will help; can always drop it later.

- **Use clustered indexes.**

  Appropriate use of clustered indexes can tremendously increase performance. Even UPDATE and DELETE operations are often accelerated by clustered indexes, because these operations require much reading. A single clustered index per table is allowed, so use this index wisely. Queries that return numerous rows or queries involving a range of values, are good candidates for acceleration by a clustered index.

  Examples:

```
SELECT * FROM PRSSEWINGTICKETINGDETAIL2
WHERE COLORCODE = 'PINK'
-or-
SELECT * FROM PRSSEWINGTICKETINGDETAIL2
WHERE  MARKERNO > 5
 AND MARKERNO < 10
```

By contrast, the COLORCODE or MARKERNO columns mentioned above are not good candidates for a non-clustered index if this type of query is common. Try to use non-clustered indexes on columns where few rows are returned.

- **Examine column uniqueness.**

  This helps to decide what column is a candidate for a clustered index, non-clustered index, or no index.

  The following is an example query to examine column uniqueness:

```
SELECT COUNT (DISTINCT COLORCODE)
FROM PRSSEWINGTICKETINGDETAIL2
```

This returns the number of unique values in the column. Compare this to the total number of rows in the table. On a 679760-row table, 339880 unique values would make the column a good candidate for a non-clustered index. On the same table, 1360 unique values would better suit a clustered index. Three unique values should not be indexed at all. Must place the indexes on the individual columns listed in the WHERE clauses of the queries.

- **Examine data distribution in indexed columns.**

  Often a long-running query occurs because a column with few unique values is indexed, or a JOIN on such a column is performed. This is a fundamental problem with the data and query itself, and cannot usually be resolved without identifying this situation. For example, a physical telephone directory sorted alphabetically on last name will not expedite looking up a person if all people in the city are named just "Smith" or "Jones." In addition to the above query, which gives a single figure for column uniqueness, can use a GROUP BY query to see the data distribution of the indexed key values. This provides a higher resolution picture of the data, and a better perspective for how the optimizer views the data.

  The following is an example query to examine data distribution of indexed key values, assuming a two-column key on COLORCODE, MARKERNO :

```
SELECT COLORCODE, MARKERNO, COUNT(*)
FROM PRSSEWINGTICKETINGDETAIL2
GROUP BY COLORCODE, MARKERNO
```

This will return one row for each key value, with a count of the instances of each value. To reduce the number of rows returned, it may be helpful to exclude some with a HAVING clause. For example, the clause

- HAVING COUNT(*) > 1

will exclude all rows which have a unique key.

The number of rows returned in a query is also an important factor in index selection. The optimizer considers a non-clustered index to cost at least one page I/O per returned row. At this rate, it quickly becomes more efficient to scan the entire table. This is another reason to restrict the size of the result set or to locate the large result with a clustered index.

Do not always equate index usage with good performance, and the reverse. If using an index always produced the best performance, the optimizer's job would be very simple - always use any available index. Actually, incorrect choice of indexed retrieval can result in very bad performance. Therefore the optimizer's task is to select indexed retrieval where it will help performance, and avoid indexed retrieval where it will hurt performance.

## 7.3 USE EFFICIENT QUERY DESIGN

Some types of queries are inherently resource intensive. This is related to fundamental database and index issues common to most relational database management systems (RDBMSs), not specifically to SQL Server. They are not inefficient, because the optimizer will implement the queries in the most efficient fashion possible. However, they are resource intensive, and the set-oriented nature of SQL may make them appear inefficient. No degree of optimizer intelligence can eliminate the inherent resource cost of these constructs. They are intrinsically costly when compared to a more simple query. Although SQL Server will use the most optimal access plan, this is limited by what is fundamentally possible.

For example:

- Large result sets
- IN, NOT IN, and OR queries
- Highly non-unique WHERE clauses
- != (not equal) comparison operators
- Certain column functions, such as SUM
- Expressions or data conversions in WHERE clause
- Local variables in WHERE clause
- Complex views with GROUP BY

Various factors may necessitate the use of some of these query constructs. The impact of these will be lessened if the optimizer can restrict the result set before applying the resource intensive portion of the query. The following are some examples.

Resource-intensive :

- SELECT SUM(COLORCODE) FROM PRSSEWINGTICKETINGDETAIL2

Less resource-intensive :

- SELECT SUM(COLORCODE) FROM PRSSEWINGTICKETING WHERE COLORCODE = 'PINK'

Resource-intensive :

- SELECT * FROM PRSSEWINGTICKETINGDETAIL2 WHERE LNAME=@VAR

Less resource-intensive :

- SELECT * FROM PRSSEWINGTICKETINGDETAIL2 WHERE LNAME=@VAR AND COLORCODE = 'PINK'

In the first example, the SUM operation cannot be accelerated with an index. Each row must be read and summed. Assuming that there is an index on the ColorCode column, the optimizer will likely use this to initially restrict the result set before applying the SUM. This can be much faster.

In the second example, the local variable is not resolved until run time. However, the optimizer cannot defer the choice of access plan until run time; it must choose at compile time. Yet at compile time, when the access plan is built, the value of @VAR is not known and consequently cannot be used as input to index selection.

The illustrated technique for improvement involves restricting the result set with an AND clause. As an alternate technique, use a stored procedure, and pass the value for @VAR as a parameter to the stored procedure.

In some cases it is best to use a group of simple queries using temp tables to store intermediate results than to use a single very complex query.

Large result sets are costly on most RDBMSs. Should try not to return a large result set to the client for final data selection by browsing. It is much more efficient to restrict the size of the result set, allowing the database system to perform the function for which it was intended. This also reduces network I/O, and makes the application more amenable to deployment across slow remote communication links. It also improves concurrency-related performance as the application scales upward to more users.

## 7.4 ORGANIZATION OF DATABASE SPACES

The role that application design plays in SQL Server performance cannot be overstated. Rather than picture the server in the dominant role, it is more accurate to picture the client as a controlling entity, and the server as a puppet of the client. SQL Server is totally under the command of the client regarding the type of queries, when they are submitted, and how results are processed. This in turn has a major effect on the type and duration of locks, amount of I/O and CPU load on the server, and hence whether performance is good or bad.

For this reason, it is important to make the correct decisions during the application design phase. However even if face a performance problem using a turnkey application where changes to the client application seem impossible, this does not change the fundamental factors which affect performance - namely that the client plays a dominant role and many performance problems cannot be resolved without making client changes.

With a well-designed application, SQL Server is capable of supporting thousands of concurrent users. With a poorly-designed application, even the most powerful server platform can bog down with just a few users.

Using the following suggestions for client application design will provide good SQL Server performance:

- Use small result sets. Retrieving needlessly large result sets (for example, thousands of rows) for browsing on the client adds CPU and network I/O load, makes the application less capable of remote use, and can limit multiuser scalability. It is better to design the application to prompt the user for sufficient input so that queries are submitted which generate modest result sets.

  Application design techniques which facilitate this include limiting the use of wildcards when building queries, mandating certain input fields, and prohibiting improvised queries.

- Use dbcancel() correctly in DB-Library applications. All applications should allow cancellation of a query in progress. No application should force the user to reboot the client computer to cancel a query. Not following this principle can lead to performance problems that cannot be resolved. When dbcancel() is used, proper care should be exercised regarding transaction level. The same issues apply to ODBC applications, if the ODBC sqlcancel() call is used.

- Always process all results to completion. Do not design an application or use a turnkey application that stops processing result rows without canceling the query. Doing so will usually lead to blocking and slow performance.

- Always implement a query timeout. Do not allow queries to run indefinitely. Make the appropriate DB-Library or ODBC calls to set a query timeout. In DB-Library, this is done with the dbsettime() call, and in ODBC with SQLSetStmtOption().

- Do not use an application development tool that does not allow explicit control over the SQL statements sent to the server. Do not use a tool that transparently generates SQL statements based on higher level objects, unless it provides crucial features such as query cancellation, query timeout, and complete transactional control. It is often not possible to maintain good performance or to resolve a performance problem if the application all by itself generates "transparent SQL," because this does not allow explicit control over transactional and locking issues which are critical to the performance picture.

- Do not intermix decision support and online transaction processing (OLTP) queries.

- Do not design an application or use a turnkey application that forces the user to reboot the client computer to cancel a query. This can cause a variety of performance problems that are difficult to resolve because of possible orphaned connections.

## 7.5 NORMALIZE LOGICAL DATABASE DESIGN

Reasonable normalization of the logical database design yields best performance. A greater number of narrow tables is characteristic of a normalized database. A lesser number of wide tables is characteristic of a denormalized database. A highly normalized database is routinely associated with complex relational joins, which can hurt performance. However, the SQL Server optimizer is very efficient at selecting rapid, efficient joins, as long as effective indexes are available.

The benefits of normalization include:

- Accelerates sorting and index creation, because tables are narrower.

- Allows more clustered indexes, because there are more tables.

- Indexes tend to be narrower and more compact.
- Fewer indexes per table, helping UPDATE performance.
- Fewer NULLs and less redundant data, increasing database compactness.
- Reduces concurrency impact of DBCC diagnostics, because the necessary table locks will affect less data.

With SQL Server, reasonable normalization often helps rather than hurts performance. As normalization increases, so do the number and complexity of joins required to retrieve data. As a rough rule of thumb, Microsoft suggests carrying on the normalization process unless this causes many queries to have four-way or greater joins.

If the logical database design is already fixed and total redesign is not feasible, it may be possible to selectively normalize a large table if analysis shows a bottleneck on this table. If access to the database is conducted through stored procedures, this schema change could take place without impacting applications. If not, it may be possible to hide the change by creating a view that looks like a single table.

## 7.6 ABSTRACT DATA ACCESS

The use of stored procedures distinguishes SQL Server from file-based databases. Stored procedures are containers for code stored on the server that allow to mix control-of-flow logic, such as If...Then statements, with SQL to retrieve, insert, update, and delete data from the database. This provides a powerful mechanism to help abstract the data access code from the underlying database structure. In addition, the SQL code in stored procedures is precompiled and can be stored in memory once the stored procedure is executed, improving performance.

I use stored procedures exclusively for all the data access in projects that use SQL Server. This allows the most flexibility for changing the structure of the table and provides a performance boost. Stored procedures can also be used in a multiuser situation to implement security because permissions to the underlying table.

I create a stored procedure for inserting datas. I use stored procedure over here because if we insert from VB, then we will not get the proper insertition time, and moreover we need time taken by the SQL Server not VB.

```
CREATE PROCEDURE populate @insertcount int as

/*  Delete any current rows */
delete from testtable

/* Loop until we have inserted the number of rows indicated by the
@insertcount parameter */

WHILE (SELECT count(*) from testtable) < @insertcount
begin

/*  Insert data. Note the dtInsert column is not identified  because the table is
set to give the field a default value  returned from the getdate() function */

insert into testtable(field1, field2, field3, field4, field5, field6, field7, field8,
field9, field10, field11, field12)
values('1', '2', '3', '4', '5', '6', '7', '8', '9', '10', '11', '12')

END
GO
```

**Figure 7.1 Functionality of Stored Procedure**

In this stored procedure, first delete any current rows in the table. Then the code loops until the number of rows in the table equals the value of the passed-in parameter, @insertcount. I determine the number of rows in the table using the count(*) SQL syntax. (I know there are probably more efficient ways to do this, but we want the query to run at some length so we can monitor it). I use database of 679760 rows and 11 columns. I create an ODBC data source called Payroll to my new SQL database.

When executing a query directly against a dedicated SQL test database, there's almost no pause. But if were executing a complex SQL query, working against a heavily used database, or perhaps working against a legacy system, I experience significant delays in the return of nearly any query. Using the RDC's QueryCompleted event, I have the ability to provide the user with feedback when the query is finished, to allow the user to continue working, or to carry out any other option that suits my environment.

As I can see, this feature is fairly straightforward. However, many client/server applications don't utilize the Remote Data control, especially if they're trying to completely abstract away the database interface from the user interface in a multitier application. But don't fear--I accomplish the same task using RDO without using the RDC.

Steps : - Start a new project named prjQueries.

- Add a form called frmQueries to the project.
- In the project, I'll need to reference Remote Data Objects by using the Project | References menu item.
- Following table specifies the controls you add to frmQueries.

| Control Type | Control Name |
|---|---|
| Frame | frmPopulate |
| Timer | timer1 |
| Text box | txtNumRows |
| Label | lblStatus |
| Label | lblTime |
| Command button | cmdPopulate |

**Table 7.1 Controls That Been Used in the Form**

When the code runs, I'll enter in the text box a value that indicates the number of rows I want to insert into the test table. Then, click the Populate button to begin the insertion. As the query executes, with each tick of the timer the program shows the status of the query, as well as the current execution time in seconds.

```
Option Explicit

' Globally declare our rdo environment and connection objects
Dim en As rdoEnvironment
Dim cn As rdoConnection

' Globally save the StartTime when the query  is executed
Dim StartTime As Date

Private Sub cmdPopulate_Click()

Dim SQL As String

' Get our rdo environment to work with.
Set en = rdoEnvironments(0)

' Open a connection using the sqltest system DSN
```

```
Set cn = en.OpenConnection(dsName:="PayRoll",_
Prompt:=rdDriverCompleteRequired)

If txtNumRows.Text <> "" Then

        ' Build our stored procedure execute statement. The number of rows
        'to insert into the database is retrieved from the text box
        SQL = "execute populate " & CLng(txtNumRows.Text)

        ' Get the start time
        StartTime = Now

        ' Also set the timer interval to 100 ms
        Timer1.Interval = 100

        ' Set query checking interval to 100 ms
        cn.AsyncCheckInterval = 100

        ' Execute the SQL Statement. Use the rdAsyncEnable parameter to
        'indicate we want to continue processing in the program even if the
        'query is not completed
        cn.Execute SQL, rdAsyncEnable

        ' Enable the timer to monitor the status of the query
        Timer1.Enabled = True

Else

        ' No value was entered, notify the user.
        MsgBox "You did not enter the number of rows to populate the
        database with."

End If

End Sub

Private Sub Timer1_Timer()

' Check to see if the query is still executing
If cn.StillExecuting = True Then

        ' Show the status
        lblStatus.Caption = "Status:  Still Executing"
```

```
        ' Show the current query time
        lblTime.Caption = "Query Time:  " &  DateDiff("s", StartTime, Now)

Else

        ' Indicate the query is done
        lblStatus.Caption = "Status:  Done!"

        ' Show the query time
        lblTime.Caption = "Query Time:  " & DateDiff("s", StartTime, Now)

        ' Disable the timer
        Timer1.Enabled = False

End If

End Sub
```

**Figure 7.2 frmQueries Code**

The key to understanding how this code works lies in the Populate stored procedure. In the cmdPopulate_Click subroutine, I create the RDO connection. Then I create the SQL statement to execute the Populate stored procedure and pass in the number of rows to be inserted (indicated by the user in the text box). Next, I set the timer interval to 100 milliseconds.

Now I'm at the heart of the functionality. Remote Data Objects provides a feature called asynchronous queries, which lets the VB program continue on its merry way while the SQL query continues to execute. The program will then periodically go out and check the status of the query. The periodicity is set by the AsyncCheckInterval property of the connection object.

In this case, I set it to 100 milliseconds. The value normally defaults to 1,000 milliseconds (one second).

Now I'm ready to execute the query, using the usual execute method of the connection object. But in this case, I'm going to provide one additional parameter--rdAsyncEnable--to indicate that this query should be run asynchronously. This parameter tells the program to continue processing even though the query might not be finished.

In order to monitor the query as it runs, I check the status of the connection object's StillExecuting property. I do so in the Timer1_Timer subroutine. Each time the timer fires, the code checks to see whether the query is still running. If it is, I simply update how long it's been running by subtracting the current time from the time when the StartTime variable was set. When the value of StillExecuting is false and the query is finished, the timer stops and I indicate that the query is finished. With a few hundred thoudsand row insert, I'll see a several-second delay.

## 7.7 TECHNIQUES TO ANALYZE SLOW PERFORMANCE

It may be tempting to address a performance problem solely by system-level server performance tuning. For example, how much memory, the type of file

- Use the 4032 trace flag according to the instructions in the SQL Server 4.2x "Troubleshooting Guide," and the SQL Server 6.0 "Transact-SQL Reference." This will allow capture of the SQL statements sent to the server in the SQL error log.

- Monitor the queries through a network analyzer such as Microsoft Network Monitor, which is part of Systems Management Server.

- For ODBC applications, use the ODBC Administrator program to select tracing of ODBC calls.

- Use a third-party client-side utility which intercepts the SQL at the DB-Library or ODBC layers. An example of this is SQL Inspector from Blue Lagoon Software.

- Use the SQLEye analysis tool provided as an example in the Microsoft TechNet CD. NOTE: SQLEye is not supported by Microsoft Technical Support.

After the slow query is isolated, do the following:

- Run the suspected slow query in isolation, using a query tool such as ISQL, and verify that it is slow. It is often best to run the query on the server computer itself using ISQL and local pipes, and redirect the output to a file. This helps eliminate complicating factors, such as network and screen I/O, and application result buffering.

- Use SET STATISTICS IO ON to examine the I/O consumed by the query. Notice the count of logical page I/Os. The optimizer's goal is to minimize I/O count. Make a record of the logical I/O count. This forms a baseline against which to measure improvement. It is often more effective to focus exclusively on the STATISTICS IO output and experiment with different query and index types than to use SET SHOWPLAN ON. Interpreting and effectively applying the output of SHOWPLAN can require some study, and can consume time that can be more effectively spent on empirical tests. If the performance problem is not fixed by these simple recommendations, then can use SHOWPLAN to more thoroughly investigate optimizer behavior.

- If the query involves a view or stored procedure, extract the query from the view or stored procedure and run it separately. This allows the access plan to change as experiment with different indexes. It also helps localize the problem to the query itself, versus how the optimizer handles views or stored procedures. If the problem is not in the query itself but only when the query is run as part of a view or stored procedure, running the query by itself will help determine this.

- Be aware of possible triggers on the involved tables that can transparently generate I/O as the trigger runs. Must remove any triggers involved in a slow query. This helps determine if the problem is in the query itself or the trigger or view, and therefore, helps direct the focus.

- Examine the indexes of the tables used by the slow query. Use the previously listed techniques to determine if these are good indexes, and change them if necessary. As a first effort, try indexing each column in WHERE clause. Often performance problems are caused by simply not having a column in the WHERE clause indexed, or by not having a useful index on such a column.

- Using the queries previously mentioned, examine the data uniqueness and distribution for each column mentioned in the WHERE clause, and especially for each indexed column. In many cases simple inspection of the query, table, indexes, and data will immediately show the problem cause. For example, performance problems are often caused by having an index on a key with only three or four unique values, or performing a JOIN on such a column, or returning an excessive number of rows to the client.

- Based on this, make any needed changes to the application, query, or indexes. Run the query again after making the change and observe any change in I/O count.

- After noting improvement, run the main application to see if overall performance is better.

Check the program for I/O or CPU-bound behavior. It is often useful to determine if a query is I/O or CPU bound. This helps focus the improvement efforts on the true bottleneck. For example, if a query is CPU bound, adding more memory to SQL Server will probably not improve performance, because more memory only improves the cache hit ratio, which in this case, is already high.

How to Examine I/O vs. CPU-bound Query Behavior :

- Use Windows NT Performance Monitor to watch I/O versus CPU activity. Watch all instances of the "% Disk Time" counter of the LogicalDisk object. Also watch the "% Total Processor Time" counter of the System object. To see valid disk performance information, must have previously turned on the Windows NT DISKPERF setting by issuing "diskperf -Y" from a command prompt, and then rebooting the system.

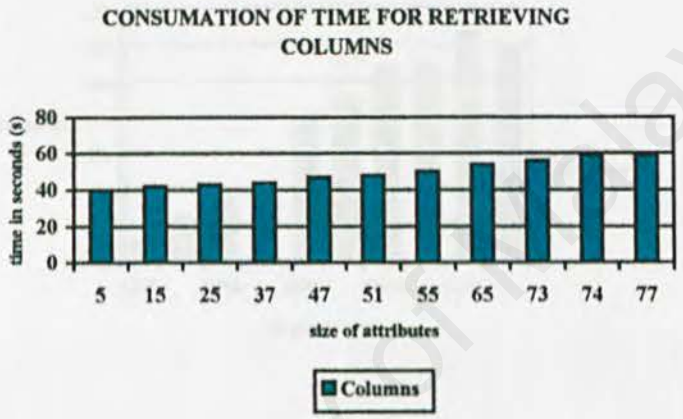
- While running the query, if the CPU graph is consistently high (for example, greater than 70 percent), and the "% Disk Time" value is consistently low, this indicates a CPU-bound state.

- While running the query, if the CPU graph is consistently low (for example, less than 50 percent), and the "% Disk Time" is consistently high, this indicates an I/O bound state.

- Compare the CPU graph with the STATISTICS IO information.

## 7.8 HIPOTHESIS

When retrieve different size of columns or attributes the time consuming is increasing. The increment in timing is not so far part and it is based on the different between the database in a table.



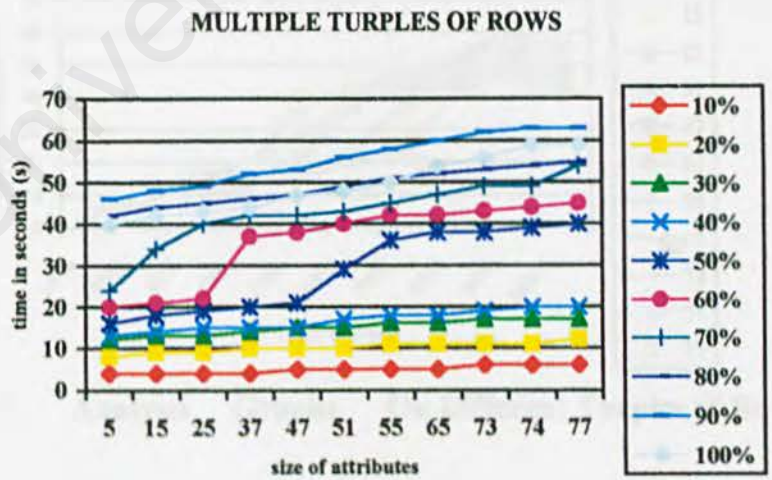
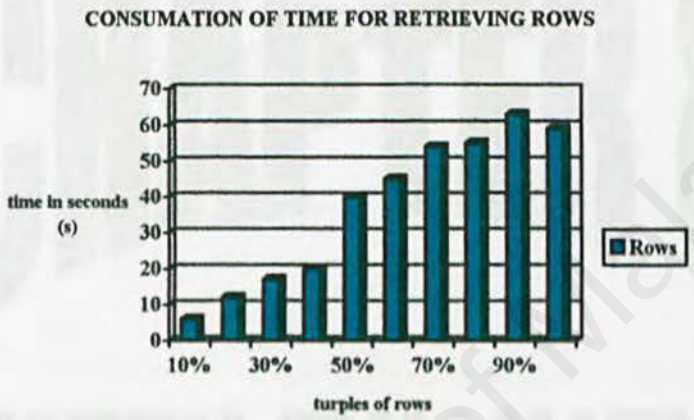




**Figure 7.3 Graph On Retrieving Different Size Of Columns**

When retrieving different turples of rows the time consuming is increasing in the earlier stage but at the 100% turple of rows it decrease compare to previous turples. This is because more time needed to sort the different rows which is not in order compare to retrieve the whole rows in order.



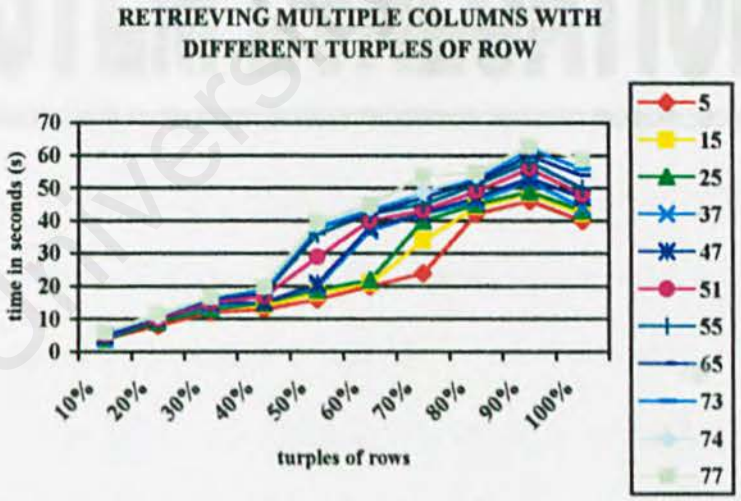




**Figure 7.4 Analysis Graphs On Different Turples of Rows**

# CHAPTER 8

# SYSTEM EVALUATION

# SYSTEM EVALUATION

## 8.1 INTRODUCTION

This chapter includes problem encountered and its solution, strengths of the system and the overall conclusion.

## 8.2 PROBLEMS ENCOUNTERED AND ITS SOLUTION

There are several problems encountered throughout the development of experiment tools system for Optimizing SQL Server including:

**a) Unsupported features between programming tools**

There are several features in SQL Server 7 that are not supported by the Visual Basic 6. The problem occurs because some of the components in Visual Basic 6 do not support the database format used by the SQL Server. For example the it can access database format of Microsoft Access. Finally, it is decided to use ADO and RDO as the system database because they can easily be converted to Visual Basic 6 with no difficulty.

**b) Programming problems**

Lack of Visual Basic 6.0 programming skills cause a lot of problems during the system coding state. However, all the problems had been solved finally by reading a lot of references and explore the Internet.

**c) Connecting to Data Sources**

There are several ways to connect to a data source. I used the the connection string method but I failed because the same errors occurs. After failed using that method, I tried by using the ODBC Data Sources or DSNs which can be created using the ODBC Data Source Administrator. I succeeded to connect to the data source by using the second method.

**d) Lack of latest references**

This is the main problem faced when finding references for the literature review and finding programming reference books. This problem had solved by visiting to University of Malaya main library, UNITEN's library and National Library.

**e) Lack Of PCs in the Lab**

The PCs available for the students were not enough in the lab. It makes me to develop my system at home and try them in the lab when situation allows.

**f) Lack of SQL Server knowledge**

Limited knowledge in SQL Server management cause the difficulty of developing an appropriate classification, designing the database and the system transactions. Guidance from supervisor and cooperation from tutors providing a clearer and better understanding of *Verification on Physical Design of Microsoft SQL Server 7.0 Performance* process and procedures.

## 8.3 EVALUATION BY ENDUSERS

The system is mainly as a system of an experiment set to analyze the optimum time to retrieve database from SQL Server by rows and columns. The system had been tried by myself to get the stimulation results for my research. I'm very satisfied with this system of an experiment set.

## 8.4 SYSTEM STRENGTHS

*Vrification on Physical Design of Microsoft SQL Server 7.0 Performance* is a research with system of an experiment set ith a good features. Its strengths are discussed below .

**1) Attractive and Simple Graphic User Interface**

The interface of the system is simple and user friendly. The pages has colours to differentiate the contents. This makes the pages easily to be viewed and not clumsy. This factor is important for the who might be a beginner . The simple looking pages can make them feel comfortable.

**2) Good Navigation**

Can navigate easily in the whole pages by using the links. The links can be found at the main page and at the left side of the pages. This feature is important to make the users move easily and use the system effectively.

**3) Good Division of Sections**

*Verification* is a time execution on retrieving the data from SQL Server which has many functions for the purpose of keeping time consuming information. They are divided into three main section according to their functions such as columns; rows; and columns and rows. The forms are arranged nicely so that won't feel difficult to execute with the forms.

**6) Standalone System**

The system is a standalone system and it makes easy to work on with it. It is very close to research system and only can work with huge datas.

## 8.5 SYSTEM LIMITATIONS

This sytem of experiment set is not a perfect system. It has a few limitations which make the system weak.

**1) Time Consuming**

The system is developed using Visual Basic 6 and it is not a good application to view time consuming because the timing is not so consistent and accurate. Example : if the time taken is 5.003 seconds and it will consider as 5 seconds only.

**2) Help Menu**

The Help menu is not been provided for the references of the use. It doesn't need Help menu because lack problems to be solved. They might have problems that has not been thinked of during developing process yet not so critical.

**3) Performance Dependent On SQL Server**

The Student's Organizer will work fine when the system is standalone. However, problems happens when the server is slow or bad upon requests from database because the performance of the system based on SQL Server.

## 8.6 FUTURE ENHANCEMENTS

In order to maintain the system and the database attractively and usability, few features can be updated or added. The system ought to be updated with more features in future.

**1) Integrate with more tables of databases**

It will be better if the system could retrieve datas from various tables. By this, it can be more appropriate tools for research purpose.

**2) Graphical Inputs**

It will be better if the output of results been collected and shown in the graphical methods. This allows to get informations gathered in graphical output after each testing.

## 8.7 KNOWLEDGE AND EXPERIENCE GAINED

Verification of SQL Server's performance is very popular and there are so many research been done regarding verification. However, the concept behind it is quite challenging and I discovered them during the development process of *Verification on Physical Design of Microsoft SQL Server 7.0 Performance*.

At the beginning of the development process, I just knew that I'm going to develop a system application. However, I had a minor understanding on the optimization concept. After reading up them, I came to know that optimization application can be created to function dynamically by using T-SQL. This makes me to learn the T-SQL and the scripting language that related to SQL. However, I don't use all this codes.

Visual Basic 6 is a development tool which can be used to retrieve datas from SQL Server. So, I started to use it and develop my system. Later, I had problem to connect to my database. I got to know that there are few ways to get connected to our data source. All this were done using ODBC and DNS which I run to view my standalone system.

As conclusion, a lot of valuable knowledge has been gained throughout the development of the system of an experiment set like Visual Basic language, SQL programming, database designing and accessing.

## 8.8 SUMMARY

System evaluation phase has made me to think of all the problems that I encountered during the system development. It is quite challenging to do project all alone where we have to think of the system thoroughly. Even though we succeeded in creating the system, it still has weaknesses and it can be overcome in future by allowing future enhancements.

# CHAPTER 9

# CONCLUSION

# THE FUTURE OF SQL SERVER

## 9.1 INTRODUCTION

The computer industry is undergoing a revolution. Between 100 and 150 million people world-wide use a PC each and every day. The growth in client server products and tools has fueled the dramatic reassessment of corporate computing requirements and evaluation of the need for mainframe-centric computing. Even so, 80% of the world's data is still resident on flat file data storage systems and the number of companies that have actually gone through the process of re-engineering their business from top to bottom and actively leverage client server technology is remarkably small.

The Microsoft company strategy is based on the decision to focus on key areas :

- Deskstop computing
- Consumer "Microsoft at Home" brand
- Information Superhighway
- Enterprise computing

## 9.2 WHAT IS THE ENTERPRISE?

Microsoft are determined to build the type of software that will run on any size of networked based personal computer. Alongside this comes a need to build additional software to compliment the existing portfolio, including transaction monitors, development tools and respositories.

But what exactly is the enterprise? The classical view is the typical mainframe based system, of which there are approximately 300,000 systems world-wide and 300,000 or so smaller mini type systems such as AS/400 and larger VAX based systems. The traditional playground for DB2, Oracle and Sybase and a weaker area for Microsoft.

The alternative view to the traditional enterprise arena is radical. Windows NT has been called a true "1/2 hour OS". This radical philosophy extends down to the hardware. The days of large single box solutions comes to an end, being replaced by a cluster of smaller servers, which grow over time to produce infinitely scaleable solutions.

But large boxes will be present for many, many years. Microsoft are focusing on producing software that can integrate into that environment in a seamless fashion. And to this over all end, Microsoft have hired 20 or so of the world's top database experts to create a truly formidable expert team.

## 9.3 DATABASE AND DEVELOPER TOOLS FUTURES

The first move in reaching the future architectural goal was to re-evaluate the product development teams. The development teams have been stream lined to focus on building a component based development environment.

The database engines are undergoing a review and work has started on creating a single unified database engine, which will be scaleable from a single processor PC to a multiprocessor server running Windows NT.

All of these components need somewhere to be stored, and this will take the form of a software repository which is currently under design with teams from Microsoft and Texas Instruments, who are generally acknowledged to be the leaders in respitory based technology.

The engineering effort to create such a complex architecture should not be underestimated, but once delivered should offer an exciting array of development possibilities.

# REFERENCE


- Homer, Alex. *Alex Homer's Professional ASP Techniques.* Birmingham, U. K. : Wrox Press, 2000

- Mortensen, Lance. *MCSE : SQL Server 7 Administration Study Guide.* San Francisco : Sybex, Network Press, 1999

- Freeze, Wayne S. *The SQL Programmer's Reference : Windows 95/NT & UNIX.* Research Triangle Part, NC : Ventana, 1998

- Watterson, Karen L. *10 Projects You Can Do With Microsoft Sql Server 7,* 2000

- Purba, Sanjiv. *Building Microsoft SQL Server 7 Applications With COM.* New York : John Wiley, 1999

- *Microsoft SQL Server 7 Database Implementation Training Kit.* Redmond, WA : Microsoft Press, 1999

- *Microsoft SQL Server 7.0 DBA Survival Guide.* Indianapolis, IN : SAMS, 1999

- Bjektich, Sharon. *Microsoft SQL Server 7.0 Unleashed.* Indianapolis, NA : SAMS, 1999.

- Dalton, Patrick. *Microsoft SQL Server Blackbook.* Albany, NY : Coriolis Group Books, 1997

- *Practical Microsoft SQL Server 7.0.* Indianapolis, IN : Que, 1999

- Wynkoop, Stephen. *Special Edition Using Microsoft SQL Server 7.0.* Indianapolis, Ind. : Que, 1999

- England, Ken, 1995. *The SQL Server Handbook : A Guide To Microsoft Database Computing.* Boston : Digital Press, 1996

- Shepker, Mathew. *Writing Stored Procedures For Microsoft SQL Server.* Indianapolis, IN : SAMS, 2000

- Rahmel, Dan. *MSDE developers guide.* Foster City, CA : M & T Books, 2000

- Coffman, Gayle. *SQL Server 7 : the complete reference.* Berkeley, Calif. : Osborne / McGraw-Hill, 1999

- *SQL Server : data warehousing*. Berkeley, Calif. : Osborne / McGraw-Hill, 1999

- Sawtell, Rick. *SQL Server 7 : 24*. San Francisco : Sybex. 1999

- Nath, Aloke. *The guide to SQL server*, 2nd ed. Reading, Mass. : Addison-Wesley Pub.Co., 1995

- Vieira, Robert. *Professional SQL server 7.0 programming*. Birmingham : Wrox Press, 1999