Database Evaluation

A Comparative Investigation and Evaluation of Oracle 9i and SQL Server 2000 with respect to Performance and Scalability

Phathisile Sibanda

November 2005

Supervisor: John Ebden

Thesis submitted in partial fulfilment of the requirements for the Bachelor of Science (Honours) Degree in Computer Science at Rhodes University

RHODES UNIVERSITY
Where leaders learn

ABSTRACT

Performance and Scalability are two supreme factors determining database availability and reliability. This is especially true for modern computer systems due to the inclusion of the Internet in Online Transaction Processing (OLTP) and E-commerce applications which use databases. This trend has been motivated by the need to accommodate many users simultaneously accessing the databases from many different networked locations. This evolution gave birth to an enduring need for high throughput, good response time, excellent data consistency and concurrency control. This project evaluates SQL Server 2000 and Oracle 9i with respect to performance and scalability. Initial performance tests showed that at low to medium workload both database systems behaved in a similar fashion in terms of throughput, response time and resource utilisation. At high load however Oracle 9i emerged superior in all cases.

ACKNOWLEDGEMENT

The road to the completion of this investigation has not been an easy one. I would like to express my utmost gratitude to anyone who honestly contributed its completion. Although you might not be explicitly mentioned here, your input was of great significance: *Thanks.*

Firstly I would like to thank God for the strength he has given me throughout the year (2005). I thank him for having given me the opportunity to work with all the people I met here at Rhodes who I must thank for being such an inspiration to me; everyone has been so supportive.

Secondly I would like to thank my brilliant supervisor Mr John Ebden whom I have great respect for. He was always ready to help and give as much support as he could despite the busy schedule he had as a lecturer.

This project could not have been possible without the assistance from the technical department people especially Chris who used to assist me with my Network Access Points which always gave me problems and for doing software downloads for us.

I also thank other Honours friends that include Christo, Fred, David, Paul and Nhamo and others not mentioned, their assistance was pivotal to the completion of this project.

I would like also to thank ANDREW MELLON PRESTIGIOUS SCHOLARSHIP for their financial support which made it possible for me to be here at Rhodes University this year, 2005, thanks very much for your support.

Lastly I would like to take this opportunity to acknowledge the financial support from Telkom SA, Business Connexion, Converse SA, Verso Technologies and THRIP through the Telkom Centre of Excellence at Rhodes University which assisted the department to maintaining and giving us good service in terms of purchasing software and hardware for us, *Thank you*.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

Introduction and Project Background Information

Chapter 1 Highlights

- Project Aim
- Motivation
    - Why performance and scalability were used for this particular investigation.
    - How database vendors fudge benchmark results.
- Overview of the project
- Overview of SQL Server 2000
- Overview of Oracle 9i
- Summary of Chapters
- Chapter Summary

CHAPTER 1 : Introduction and Project Background Information

1.1 Project Aim

The primary aim of this project is to investigate and evaluate, through a sequence of performance and scalability tests, which of the two Database Management Systems (Oracle 9i or SQL Server 2000) performs and scales better than the other under certain specified conditions. For this project the author only considered performance and scalability of the DBMS, other aspects like database security, integrity and the recovery system were deliberately ignored and thus treated as constant. In addition, the output of this evaluation would be valid only for the specified scenarios. This is largely because comparing performance and scalability of two DBMS is a difficult and complex process mainly due to the fact that in most cases database performance depends on the experience of the Database Administrator (DBA) to fine-tune his database and other factors such as network connectivity and hardware and software configurations at the time of the benchmarking.

The secondary goal of the project would be to determine why the DBMS systems reacted the way they did during the testing process by performing a technical comparison of the two database systems. Factors such as indexing, partitioning, parallel execution will be investigated and compared.

The author aims to learn and appreciate performance benchmarking techniques including the processes of performance and scalability testing together with their related methods that comprise functional, load and stress testing. Furthermore the author seeks to gain knowledge of how relational databases operate as well as understand how they influence provision of services in this database dependent era.

1.2 Project Motivation

There is a strong debate amongst Database Administrators (DBA) as to which DBMS to choose under given circumstances mainly because of two main facts:

1. The growing number of open source databases flooding the market.
2. The inclusion of the Internet in database systems especially in Online Transaction Processing.

The first point has impelled serious completion between commercial and open source database vendors while the second has resulted in an enduring need for high throughput, good response times, good data consistency and concurrency to enable quick data access and servicing of requests. This whole controversy has in the past resulted in database vendors flawing performance benchmark results as explained under the section “How database vendors fudge benchmark results” below in a bid to falsely win more clients. This project seeks to assist and insulate Database Administrators by providing them with a third voice and guideline they can use to evaluate database performance and scalability before they make their choices since speed has become one of the vital factors affecting database availability.

According to Hansen [1996, pg 432] the procedure for selecting a DBMS is based on one or a combination of the following criteria:

- Response Time requirements: Response time could be the critical factor in some cases as I have already pointed out above for Internet dependent systems. Modern database systems are web-oriented thus require fast processing of requests since the systems are accessed by millions of concurrent users. However response time might be trivial for standalone database systems servicing fewer users.
- Maintaining data consistency: This category involve issues such as integrity, security and recoverability. Databases like Military DBMS are high security risk systems therefore must be more secure than Banking DBMS systems which hold integrity and recoverability at high priority.
- Application requirements: This largely depends on who the user is and what are their information requirements.

The evaluation performed for this project was based on the first mentioned criterion, since performance and scalability both are dependent on response time.

1.2.1 Why performance and scalability?

Performance refers to the ability to retrieve information within a reasonable time and at a reasonable cost [Rob *et al*, 2002, pg 448].Database performance is influenced by factors such as communication speeds, number of concurrent users ,resource limitations, system (memory and CPU type) and database configuration parameters (access path definition ,use of indexes and buffer size – Oracle 9i).Scalability on the other hand is the system ability to process more workload, with a proportional increase in system resource usage. In other words, in a scalable system, if you double the workload then the system should use twice as many system resources. This notion could be proven other wise due to system conflicts.

Performance and scalability are good factors for evaluating databases since they are the most likely measures to be used in future for the determination of database availability and reliability. This is due to the spiralling enlargement of computer systems communicating over the internet. How fast your system is in terms of servicing requests determines how many concurrent clients are able to log on to the server and do meaningful work thus making maximum use of the expensive computer hardware. Other evaluation factors that could have been used include:

- Security
- Integrity
- Backup and the Recovery System
- Ease of use and Cost

1.2.2 How database vendors fudge benchmark results

Database vendors employ numerous tricks to improve the processing speed of their DBMS systems during benchmarking and thus falsely prove that their database product is superior to that of its competitors. Almost all their tricks involve caching data and SQL statements in RAM prior to tests. These tricks are possible due to the fact that database performance is all about disk I/O, and vendors use large RAM memory regions to preload critical benchmark components to avoid any disk accesses [Burleson, 2002].

Some of their tricks include:

- **Buffering up data rows** - by preloading the data into the RAM buffers, the database can access information thousands of times faster than a disk I/O access.

- **Storing SQL execution plans in RAM** - by pre-parsing and pre-computing the execution plans for the SQL, the database vendors bypass the overhead of parsing and invoking the SQL optimizer to generate the execution plan [Figure 1.1 below].

- **Pre-joining tables** - some database products have special pre-aggregation mechanisms to pre-join tables. For example, Oracle uses Materialized Views that can store the results of an n-way table join, allowing super-fast data access.

- **Using high-speed CPUs and clusters** - Database vendors can dramatically improve benchmark speeds by using special high-speed machines and cluster architectures. For example SQL server 2000 support federated databases which allow Query broadcasting leading to high speed SQL operations.

Figure 1.1 below shows the correct execution paths of a query by the two database systems.

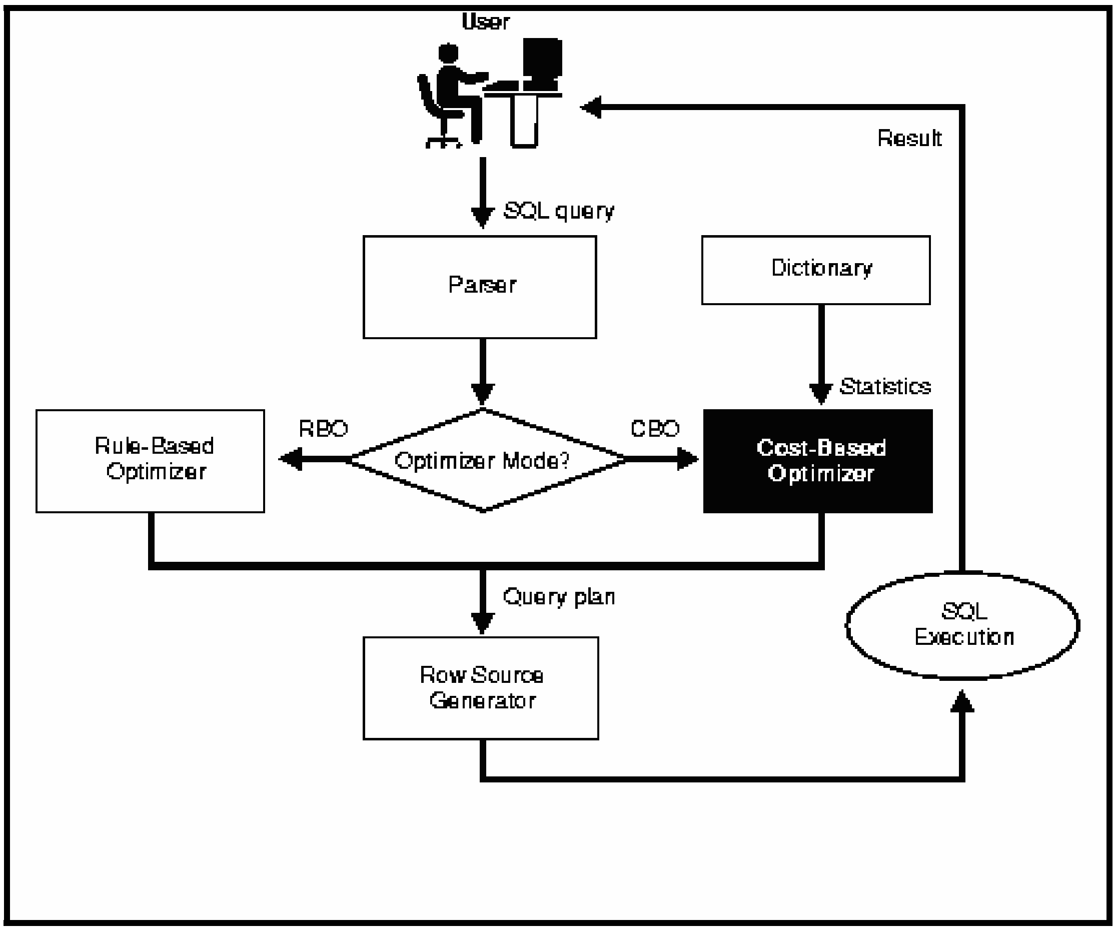


Figure 1-1 *SQL Processing Architecture*

Figure 1.1 shows that under normal circumstances, if the execution path of a SQL statement is followed correctly, the parser has to invoke the Optimizers (Rule-Based and Cost- Based Optimizers) before getting to the Row Source Generator. This is a CPU intensive task which uses a sizable number of CPU cycles thus by bypassing this step Database Vendors save optimiser execution time and thus compromise the benchmarking process and prove their systems to be faster than otherwise would be the case. All these tricks reduce disk I/O overhead by accessing all required information to process requests from RAM thus greatly improving the overall database performance. Therefore most database benchmark data should be viewed with healthy scepticism [Burleson, 2002].

This project therefore seeks to produce a third party voice for DBA just to complement vendor benchmarks.

1.3 Overview of the project

This project is divided into three sections. The first of these sections gives detailed information about the considerations and methodologies used for the design of experiments. The second is the implementation of the performance and scalability tests. This section also present results for analysis and comparison. The last section concentrates on the analysis of obtained by evaluating some architectural factors that could have influenced performance and scalability tests.

The performance testing section is further divided into three groups comprising of the functional testing, load testing and stress testing. Functional testing was done prior to any tests in a bid to create the DBMS regimen (DBMS operational baseline/schedule) which was used to evaluate the systems in subsequent tests. Having compiled the baseline the next step was to perform the load testing which was done using common database tools such as System Monitor, SQL Server 2000 Profiler, SQL Server 2000 and Oracle 9i Enterprise Managers and Statspack for Oracle 9i together with third party software such as Quest Central for data generation, load simulation and data capture. Results obtained from the load testing were used as input to the stress testing process. Typically to stress test the systems the author needed to obtain the extreme load at which the systems performed badly from the load testing phase. Stress testing follows load testing then scalability testing concludes this section.

The third section investigated the following technical factors: the concurrency model, indexing, partitioning and parallel execution capabilities.

1.4 Overview of SQL Server 2000

SQL Server 2000 is an enterprise-class proprietary Relational Database Management System from the Microsoft Corporation. From the Microsoft website it is clamed that SQL Server 2000 exceeds dependability requirements and provides innovative capabilities that increase business effectiveness, integrate heterogeneous IT ecosystems, and maximize capital and operating budgets. It is a database that has been benchmarked for scalability, speed, and performance providing core support for Extensible Mark-up Language (XML) and Internet queries.

This short description from Microsoft shows that SQL Server 2000 is definitely one of the de facto databases in the market today which commands a reasonable percentage of the market share as illustrated in figure 1-2 below. Their dominancy in the market place is one of the reasons why SQL Server 2000 (78% usage) and Oracle 9i (55% usage) were selected for this evaluation.

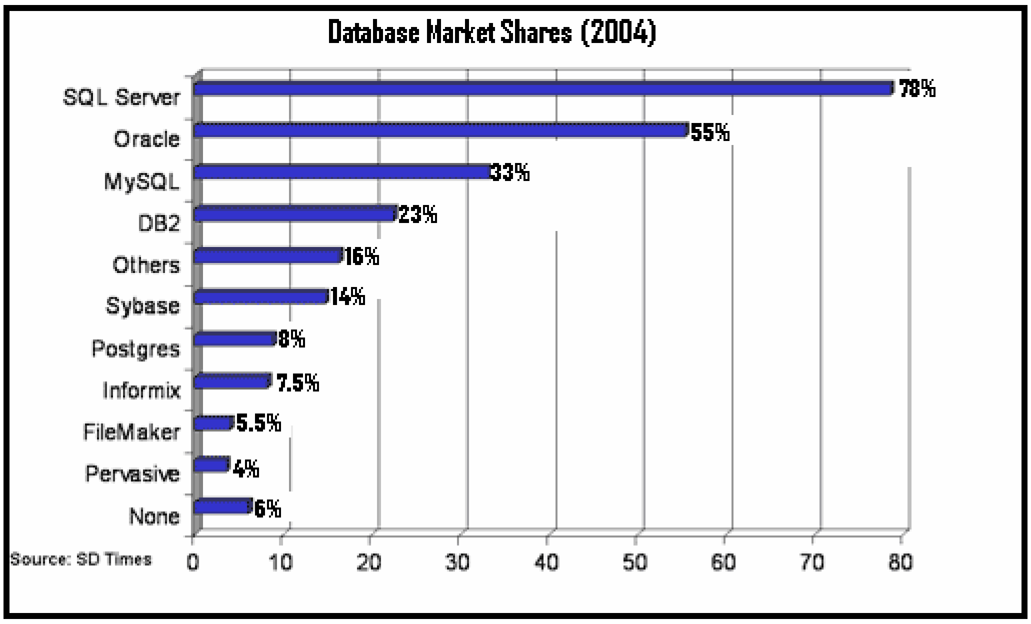


Figure 1-2 *Diagram shows Market Shares for popular DBMS systems*

Source: SD Times http://www.mysql.com/why-mysql/marketshare/

These percentage figures were calculated from the number of customers each database polled in a usage survey conducted in July 2004 by SD Times.

1.5 Overview of Oracle 9i

If Microsoft SQL Server 2000 is one of the biggest databases on the market today then Oracle 9i from the Oracle Corporation is its biggest competitor [Figure 1-2 illustrates this]. Oracle 9i is also a proprietary Relational Database Management System just like SQL Server 2000. According to Oracle database experts Oracle 9i is the really enterprise solution given the functionality it offers. Some of the technical aspects it provides include database performance and scalability enhancements, ease of management, security and availability. Additional features originally specific to Windows 2000 integration and application areas comprise of Internet content management, E-commerce integration, packaged applications and business Intelligence. Its growing market dominancy influenced its selection for this evaluation.

1.6 Overview of project chapters

Chapter 2:-This chapter gives the experiment design issues in detail. It gives the machine and software configuration information used to perform the tests. Details of the Functional test; Performance and Scalability experiment design aspects are also explained.

Chapter 3: - This chapter gives the implementation details of the tests and presents the results on various line graphs which include: Throughput Vs Number of Users, Response Time Vs Number of Users and Resource Usage Vs User Load.

Chapter 4: - This chapter discusses and analyses results from the preceding chapter and presents a technical comparison of the two systems in which the author investigated factors most likely to have caused the DBMS to behave the way they did in the tests.

Chapter 5: -This chapter concludes the project by giving the conclusion and stating the possible future research work in relation to this project.

1.7 Chapter Summary

This chapter is devoted to introducing the project and giving background information for the reader. The aim of the project is given in the first few paragraphs before a detailed explanation of why this project was chosen. Specifically the motivation expands on why “performance and scalability” were used for this investigation including a brief list of other aspects that could have been used for this experiment. Mentioned factors include security, integrity, backup and recovery System. , Ease of use and the Cost of operating and maintaining these DBMS systems. A brief overview of the project, Oracle 9i, SQL Server 2000 and chapters is given just to provide the user with a quick reference and starting point. Chapter 2 explains in detail the design issues used for the testing experiments.

Chapter 2

Design of the Experiments

Chapter Highlights

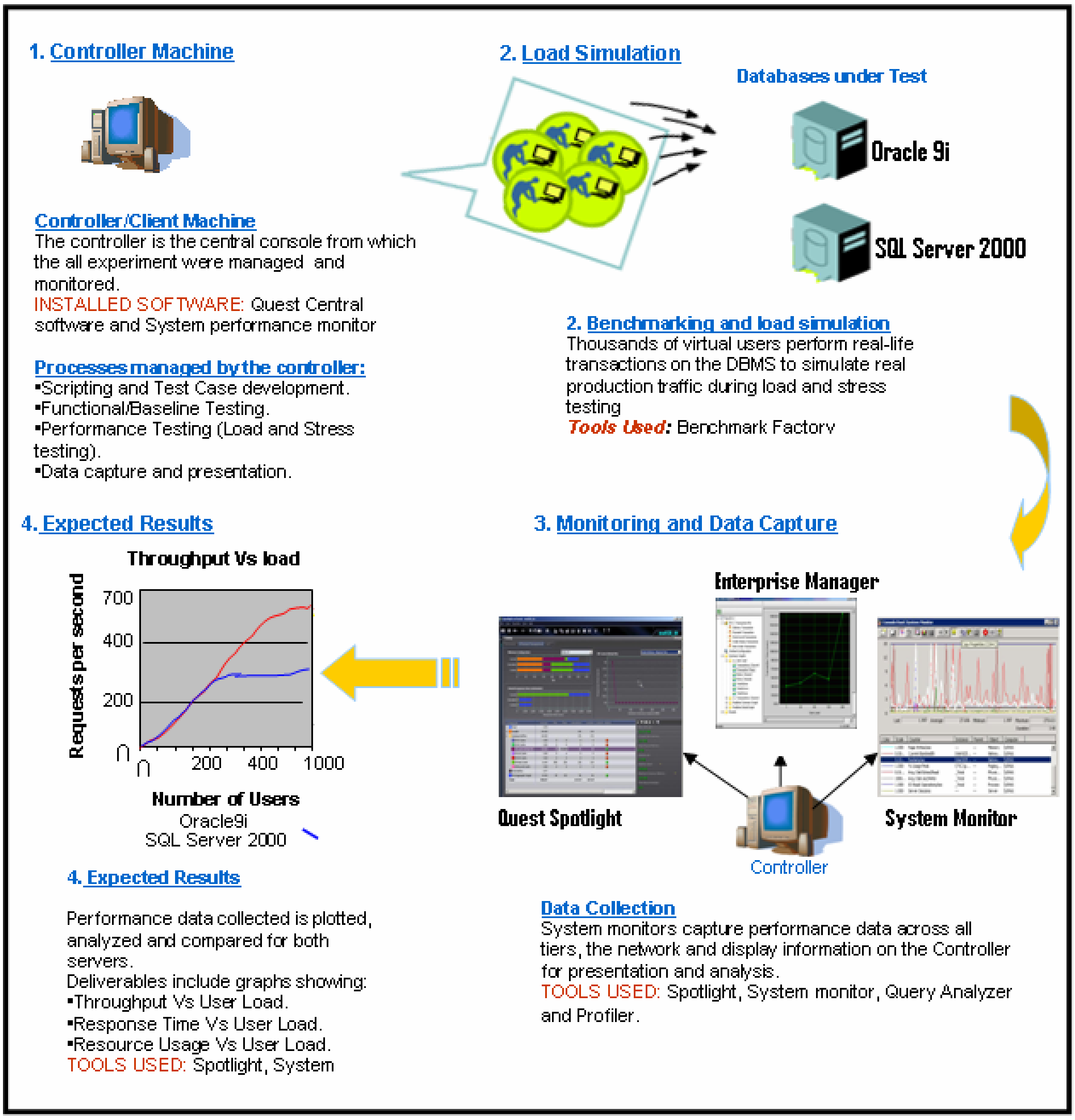
- Use Case Diagram and Descriptions
- The Performance and Scalability Testing Environment
- Functional Test/Baseline Test
- Performance Tests
    - Load Testing
    - Stress Testing
- Scalability Testing
- Data Capture and Presentation

CHAPTER 2 : Design of the Experiments

This chapter discusses the considerations used for the design of experiments in order to ensure that measurements were correctly taken. It starts by defining the testing environment and then presents details of the performance and scalability testing processes.

2.1 The Performance and Scalability Testing Environment

The testing environment comprised of three machines as depicted by Figure 2-1 below.



Controller/Client Machine
The controller is the central console from which the all experiment were managed and monitored.
INSTALLED SOFTWARE: Quest Central software and System performance monitor

Processes managed by the controller:
•Scripting and Test Case development.
•Functional/Baseline Testing.
•Performance Testing (Load and Stress testing).
•Data capture and presentation.

2. Benchmarking and load simulation
Thousands of virtual users perform real-life transactions on the DBMS to simulate real production traffic during load and stress testing
Tools Used: Benchmark Factory

4. Expected Results

Performance data collected is plotted, analyzed and compared for both servers.
Deliverables include graphs showing:
•Throughput Vs User Load.
•Response Time Vs User Load.
•Resource Usage Vs User Load.
TOOLS USED: Spotlight, System

Data Collection
System monitors capture performance data across all tiers, the network and display information on the Controller for presentation and analysis.
TOOLS USED: Spotlight, System monitor, Query Analyzer and Profiler.

Figure 2-1 *Diagram shows the Performance Testing Environment (Project Overview)*

The client machine was used as the controller of all experiments. This machine was the focal point from which both servers were remotely managed, monitored and controlled. Load and data simulation was remotely performed from the client machine during the testing process using TPC-C[1] benchmark embedded in Quest Central software. In addition this machine was also used for capturing activity data on the servers for presentation and analysis. All three machines, the controller machine, Oracle 9i and SQL Server 2000 servers were connected to each other through a 100 megabytes connection to the Rhodes LAN in the Computer Science department [Figure 2-1].

The kind of set up shown in Figure 2-1 meant that performance test results were to be influenced by network activity. To counter this effect the author made sure that experiments were performed during periods of low network activity, for instance at night when there is less network traffic. Moreover the measured results which included a network latency component were further resolved by subtracting the network delay component to get the correct performance measure which excluded network activity. Moreover, remote monitoring of performance and scalability tests was not done for all scenarios in which case network sensitivity was not an issue since testing was done on the actual DBMS system.

2.1.1 Machine Configurations

The following series of tables show machine configurations used for each machine during the testing process.

Machine Name	Machine/CPU	Memory	Network	Disk Space	Software
Client Machine (Hons11)	**Intel(R) Pentium(R) 4 CPU 3 GHz * 2 Processors**	**1 GB of RAM**	**10 Mb Ethernet Connection**	**40 GB**	**Microsoft Windows XP Professional, Version 2002, Service Pack 2**

Table 2-1 *Client Machine configuration properties*

Hons11 – is the name of the controller machine.

Ora1 – (Table 2-2) is the of the Oracle 9i server.
SS1 – (Table 2-3) is the name of the SQL Server 2000 server.

[1] TPC-C stands for .Transaction Processing Performance Council Version C for Online processing [Transaction Processing Performance Council, 2005].

Machine Name	Machine/CPU	Memory	Network	Disk Space	Software
Oracle 9i (ora1)	**Intel(R) Pentium(R) 4 CPU 2.80 GHz * 2 Processors**	**1 GB of RAM**	**100 Mb Ethernet Connection**	**100 GB hard Drive**	**Microsoft windows Server 2003. Standard Edition. Service Pack 1**

Table 2-2 *Oracle 9i configuration informantion*

Machine Name	Machine/CPU	Memory	Network	Disk Space	Software
SQL Server 2000 (ss1)	**Intel(R) Pentium(R) 4 CPU 2.80 GHz * 2 Processors**	**1 GB of RAM**	**100 Mb Ethernet Connection**	**35 GB hard Drive**	**Microsoft windows Server 2003. Standard Edition. Service Pack 1.**

Table 2-3 *SQL Server 2000 configuration information*

Oracle 9i documentation outlines best practices that should be followed when doing preparations for benchmarking and in accordance with this project I list below a few of the guidelines which I considered prior to performance testing:

- ***You should test with realistic data volumes and distributions:*** All testing must be done with fully populated tables. The test database should contain data representative of the production system in terms of data volume and cardinality between tables. All the production indexes should be built and the schema (for Oracle 9i) statistics should be populated correctly.

- ***Use the correct optimizer mode:*** All testing should be performed with the optimizer mode that will be used in production. Oracle recommends the cost-based optimizer which by default was the one used for this project.

- ***Test a single user performance:*** A single user on an idle or lightly used system should be tested for acceptable performance (this criterion was fulfilled during Functional testing). If a single user cannot get acceptable performance under ideal conditions, it is impossible there will be good performance under multiple users where resources are shared.

- ***Obtain and document plans for all SQL statements:*** Obtain an execution plan for each SQL statement, and some metrics should be obtained for at least one execution of the statement. This process should be used to validate that a good execution plan is being obtained by the optimizer and the relative cost of the SQL statement is understood in terms of CPU time and physical I/Os. This process assists in identifying the heavy use transactions that will require the most tuning and performance work.

- ***Attempt multi-user testing:*** This process is difficult to perform accurately, because user workload and profiles might not be fully quantified. However, transactions performing DML statements should be tested to ensure that there are no locking conflicts or serialization problems. There are a lot of free software in the market today that can do load simulation for you. For this project the author used Benchmark Factory included in Quest Central for Oracle and SQL Server.

- ***Test with the correct hardware configuration:*** It is important to test with a configuration as close to the production system as possible. This is particularly important with respect to network latencies, I/O sub-system bandwidth and processor type and speed. A failure to do this could result in an incorrect analysis of potential performance problems [Tables 2-1 through Table 2-3 show configurations used for this project].

- ***Lastly measure steady state performance:*** When benchmarking, it is important to measure the performance under steady state conditions. Each benchmark run should have a ramp-up phase, where users are connected to the application and gradually start performing work on the application. This process allows for frequently cached data to be initialized into the cache and single execution operations, such as parsing, to be completed prior to the steady state condition. Likewise, at the end of a benchmark run, there should be a ramp-down period, where resources are freed from the system and users cease work and disconnect.

These guidelines were strictly followed throughout the tests from the preparation phase to the completion of tests.

2.2 Functional Test/Baseline Design

2.2.1 Baseline Definition

A baseline is simply "a set of measurements that tell you how a server behaves while at rest". In more practical terms, a baseline is a complete record of a system performance taken immediately after the server becomes operational, but before the server is put under any measurable load. This baseline is taken in a controlled environment, so that external activities (network sensitivity for instance) do not impact the baseline measurements in a bid to collect accurate results. All significant features of system performance are measured against predefined metrics[2] and are later recorded and analyzed [Kevin Kline, 2005].

Functional testing goals include:

- Telling you all about the performance of a server under normal conditions for later comparison with performance and scalability test results.
- To help you document and understand as many as possible background processes running on the server which assists you in recognising different processes during testing. A good example includes separation of performance test processes from server processes.
- Assist you in building filters to catch "do not respond" situations before performance testing starts. This allows you to respond to problems that need to be rectified before stressing your system.

2.2.2 Functional Test Tools

For this process the author used System Monitor to capture statistical information since it allows you to view real-time server activity and further save log statistics for future analysis. However Quest Spotlight for both SQL Server 2000 and Oracle 9i was used for advanced analysis since it provides a better graphical interface which offers easy real-time analysis but does not enable you to save statistics for later analysis like System Monitor.

2.3 Performance Tests

The performance and scalability testing methodology and definitions used in this project were taken from MSDN library [SQL Server Developer Centre, 2005]

2.3.1 Load Testing

Load testing allows you to verify that your database can meet vendor desired performance objectives which are specified in a service level agreement of the product [Meier *et al*, 1994]. RPM Solutions Pty Ltd (2004) defines load testing as tests that determine the system behaviour under various workloads which include normal and critical loads. Its main objective is to determine how

[2] Predefined metrics are simply performance objectives released with the product as certification (User Agreement) information. For example CPU utilisation for Microsoft SQL Server 2000 should not exceed 80% under any circumstances. Exceeding this threshold is a violation of certification agreement, in other words it can not be guaranteed that the system will behave as expected at that level.

system components react as the workload is gradually increased. The latter definition is the one that the author used for load testing.

Prior to any performance testing, performance objectives, or aims need to be defined and known. However since this project seeks to compare two systems as an independent voice, tests were conducted in an exploratory manner. Load testing was performed to compare the systems in three areas which include response time, throughput rates and resource utilization (CPU, network latency, disk I/O subsystem and memory usage). These were chosen because response time and throughput objectives affect database performance from the user perspective while resource utilization and workload affect database scalability.

2.3.1.1 Response time

Response time, or latency, is defined as the amount of time that is required by the server to respond to a request[3]. It can be measured from either the server side or the client side, although these approaches measure slightly different aspects of response time [Meier *et al,* 1994].

At the server side (Oracle 9i or SQL server 2000), latency is measured as the time it takes the server to finish executing the request. This is essentially the time that elapses between receiving the request and sending the response and does not include any network latency that may occur during the test. This is the setup which was used to evaluate resource utilisation by the systems.
Latency measured at the client side is the time that elapses between sending a request, and receiving the response. This includes request queue time, execution time and any network latency that may occur. There are two common measurements used with client side latency: “Time to First Byte” (TTFB), which is measured as the time between sending the request and receiving the first byte of the response, and “Time to last byte” (TTLB), which is measured as the time between sending the request and receiving the last byte of the response [Meier *et al,* 1994]. This kind of setup was used to evaluate the systems with respect to throughput and response time.

2.3.1.2 Throughput rate

The number of requests that the database server can handle per unit of time is known as throughput. This varies depending on how many users are currently and concurrently using the server, and what each user is currently doing. It is usually measured as requests per second, but transactions per second or orders per second can also be used [Meier *et al,* 1994]. For this project I use requests per second.

[3] Requests in this perspective are mainly PL/SQL query send from the client machine to the server

2.3.1.3 Resource Utilization

Server and network resource costs must be monitored in order to ensure that the servers do not consume all of the available resources. The primary resources to be monitored as stated above are: CPU usage, memory utilisation, disk I/O subsystem and network I/O latency. The resource cost of each operation can be identified, indicating which transactions use most of the resources and where the database administrators (DBA) need to focus their attention if over-utilisation occurs. The required resources for a given workload can also be determined, indicating whether the server has enough resources for the anticipated number of users [Meier *et al*, 1994]. Resource utilisation was especially important for stress testing which involved staving the system for a resource for example by submitting requests to a server while playing a movie. This means the request will have to compete with the movie for CPU cycle which as the load increases will eventually choke the system.

2.3.1.4 Load Testing Steps

Load testing is used to determine how the application will respond with various user loads. The number of users being simulated is gradually increased until one of the stated objectives is violated. For example, you may have stated as one of your objectives that the CPU usage of the server may not exceed 75%. When load testing this server, the number of users being simulated will be increased until the server's CPU usage is greater than 75%, assuming that no other objective has been violated before this occurs. By doing this, the application's maximum operation capacity can be determined, as well as any bottlenecks that may be limiting this capacity [Meier *et al,* 1994].

Load testing process consists of six steps:

i. Identification of key scenarios: key scenarios are those scenarios that are critical for performance.
ii. Identification of workload: the total application load is distributed among the scenarios identified in the first step.
iii. Identification of metrics: This is where the various metrics that will be collected are identified.
iv. Create test cases: The various test cases are identified and created.
v. Simulate load: The various loads are simulated using a load testing tool (Quest Central software), performance metrics are also collected.
vi. Analyse results: Analyse the data collected during load testing.

2.3.1.4.1 Identify key scenarios

A scenario is a set of actions that the client is likely to perform once he or she is logged on to the database. For this project the author chose those user paths which either have significant performance costs /impacts, or are resource intensive.

Scenarios which were used for this project include:

- Querying the database for certain information.
- Editing and updating a record of a database.
- Deleting a record(s) from the database.

2.3.1.4.2 Workload modelling

Workload is defined as the number of concurrent or simultaneous users accessing the server. Concurrent users are users that send request to the database at exactly the same time; they basically fire requests at the same time while simultaneous users maintain active connections to the database although they might not be doing meaningful work. Simultaneous users simulate more realistic production traffic, as users do not usually start using the system at exactly the same time. However, concurrent users are more useful when stress testing the server as used in this project

Performance characteristics or workload for each of the above listed scenario should be identified. To do this, the following factors were considered:

- The number of users for each scenario.
- The rate of requests: the number of requests received from all simultaneous and concurrent users for a certain period of time.
- The pattern of requests: the rate of requests that individual functions of the servers experienced.

After creating a workload model, begin load testing with a total number of users distributed against your user profile, and then start to increment the load for each test cycle. Continue to increase the load, and record the behaviour until you reach the threshold for the resources identified in your performance objectives.

2.3.1.4.3 Identify Metrics

Metrics can help determine if there are any bottlenecks in the database, and whether the application is achieving its objectives. Although the required metrics vary depending on the database operations, there are a few that should be collected for every test.
Various system metrics must be monitored with respect to the client machine, which is the machine

running the load test software[4]. The following table shows important counters measured for both the client machine and individual servers.

Object	Performance Counter	Description
Processor	% Processor Time/_Total	% Processor Time is the percentage of elapsed time that the processor spends to execute a non-Idle thread. It is calculated by measuring the duration of the idle thread is active in the sample interval, and subtracting that time from interval duration. This counter is the primary indicator of processor activity, and displays the average percentage of busy time observed during the sample interval. It is calculated by monitoring the time that the service is inactive and subtracting that value from 100%.
Memory	Available Bytes	Available Bytes is the amount of physical memory, in bytes, immediately available for allocation to a process or for system use. It is equal to the sum of memory assigned to the standby (cached), free and zero page lists.
	Page Reads/sec	Page Reads/sec is the rate at which the disk was read to resolve hard page faults. It shows the number of reads operations, without regard to the number of pages retrieved in each operation. Hard page faults occur when a process references a page in virtual memory that is not in working set or elsewhere in physical memory, and must be retrieved from disk. This counter is a primary indicator of the kinds of faults that cause system-wide delays. It includes read operations to satisfy faults in the file system cache (usually requested by applications) and in non-cached mapped memory files. Compare the value of Memory\\Pages Reads/sec to the value of Memory\\Pages Input/sec to determine the average number of pages read during each operation.
Network Interface	Bytes Total/sec	Bytes Total/sec is the rate at which bytes are sent and received over each network adapter, including framing characters. Network Interface\\Bytes Received/sec is a sum of Network Interface\\Bytes Received/sec and Network Interface\\Bytes Sent/sec.
Disk I/O system	% Disk Time	% Disk Time is the percentage of elapsed time that the selected disk drive was busy servicing read or writes requests.
	Avg. Disk Bytes/Transfer	Avg. Disk Bytes/Transfer is the average number of bytes transferred to or from the disk during write or read operations.

Table 2-4 *Performance counters collected by all machines [APPENDIX E]*

These counters indicate how well the client machine and the servers are performing and identify whether any bottlenecks are occurring, which may affect the final results of the tests. Appendix E gives a full list of all metrics used in this project.

2.3.1.4.4 Test Cases

A Test plan, consisting of many test cases, needs to be documented for each workload pattern identified in the "Identify Workload" step above. Test cases list information about each test to be

[4] This client machine was my PC running Quest Central, Spotlight and Benchmark factory. It was the experiment controller, this is where all load simulation, data capture and analysis was done for some tests.

run. They contain details about the number of users in the test, the duration of the test, any think time included in the test and user profiles of the test. These profiles describe the distribution of the users across the various functions of the system under test. Users are distributed according to the workload identified in the "Workload modelling" step discussed above.

Also listed here will be the expected results of each test case, which are derived from the stated objectives of the application. Expected results for the following must be stated: throughput, requests executing, average response time, and resource utilisation thresholds.

2.3.1.4.5 Load and data simulation

Using load testing tools such as Benchmark Factory and Quest Central software, tests were created for the identified scenarios and ran against the servers at the specified workload, collecting the required metrics for each test performed. The user load should then be incrementally increased for each test until one of the stated objectives is violated or until the system reaches its breaking point which could be identified by "server busy" massages or abnormal CPU usage.

2.3.1.4.6 Analyse results

The capture and analysis of results is any integral part of the benchmarking process testing thus a separate section below was solely dedicated to detailing how the author compiled data for analysis. See the section entitled "Data capture and presentation".

2.3.2 Stress Testing

Stress testing is a kind of performance testing where the server is subjected to very high loads well over its threshold, while denying it the resources required for processing that load. A good example used for this research work was removing the random think time introduced in simulating simultaneous users from the load testing. This meant that the servers had to deal with concurrent users who fired requests to the servers at the same time. This exceedingly overloaded the server since no resource additions were made to help the system. All users had to compete for system resources, for instance CPU cycles and memory, which heavily impacted system scalability and performance. Pushing the system way over its capacity in stress testing unearthed numerous bugs which the author monitored to gauge server breaking point which was used for scalability comparison in terms of how many users were able to do meaningful work before the system failed. Some of the system bugs monitored for this exercise included:

- Synchronisation issues between request (SQL Server 2000 – shared locks)
- Race conditions
- "Server busy" error messages

- Loss of data during concurrent access of the databases

2.3.2.1 Stress Testing aspects in detail

2.3.2.1.1 Synchronisation issues

Synchronisation issues are mostly a result of database applications/processes failing to communicate correctly due to errors resulting in applications/processes crashing. SQL Server crashed due to shared-locks contention between users.

2.3.2.1.2 Race conditions

Race condition occurs when a couple of processes that were in contention for a common data structure, mainly through a software coding error (under pressure at high load) in one of the database application processes, and both are able to get write access to a data structure at the same time. This corruption leads to the involved processes getting into an infinite loop and spinning for ever. Since database tables are inherently flat files they are prone to corruption. There is no inherent locking mechanism that detects when a file is being used or modified, and so this has to be done on the script level. Even if care is taken to lock and unlock the file on each access, a busy script (during multithreaded access and modification by many simulated users) can cause a "race condition" and it is possible for a file to be wiped clean by two or more processes that are fighting for the lock; the timing of file locks becomes more and more important as the user numbers increase This problem was most seen on SQL Server 2000.

2.3.2.1.3 Server busy" error messages

A memory (or resource) leak in databases occurs when the database processes lose the ability to free the memory they consume. Repeated memory leaks cause the memory usage of a process to escalate without bounds. Memory leaks are a serious problem, if the processes eat up all the memory available finally resulting in the machine stopping and not responding to instruction. “Server busy” error messages were used to identify this phenomenon.

2.3.2.1.4 Loss of data during concurrent access of the database

All the above problems can result in data losses when the database applications/processes fail to communicate correctly due to synchronisation errors during race conditions. Data can be lost during INSERT and MODIFY operations or when the data is being stored back to the database after modifications.

2.3.2.2 Stress Testing Process

The stress testing process basically followed the same six steps listed for load testing but with different content for some of them. Below the author outlines those steps that were distinctly different in content from those for load testing otherwise all those similar were not repeated here.

For stress testing to be feasible, instead of having a list of possible scenarios one needs to select one particular case to stress test. It is also possible to have a combination of scenarios for which you have performance objectives. The stress testing process requires you to obtain peak load capacity statistics from the load testing process which will serve as input.

Another important point to recognise about stress testing is that to make the process effective, the tests were carried out on the systems which already had resource intensive processes running on them. A movie was played while testing. What this means is that the stress testing process had to compete with other running processes for resources thus leading to contention and eventually starvation. It is at this stage that system bugs identified above were revealed.

2.3.2.2.1 Identification of key scenarios

This step involves selecting the scenario or multiple scenarios that you need to stress test for identifying a particular potential performance issue based on the following criterion.

- Scenarios could be selected based on how critical they are to overall server performance.
- Important scenarios are those that are most likely to affect performance. Such operations include those that perform intensive locking and synchronization, long transactions and disk-intensive I/O operations.

One such scenario that satisfies the above guidelines which the author used involved a scenario that queries the database and scans all data table to return large amounts of data. This case involves a lost of disk I/O operations, intensive use of memory thus is most likely to impact total system performance.

2.3.2.2.2 Load Simulation

Load simulation for stress testing was performed using threaded scripts written in C sharp. Threaded user requests were fired at the same time from a webpage on the controller machine to the databases. The Unified Modelling Language (UML) Use Case diagram below shows the stress testing process.

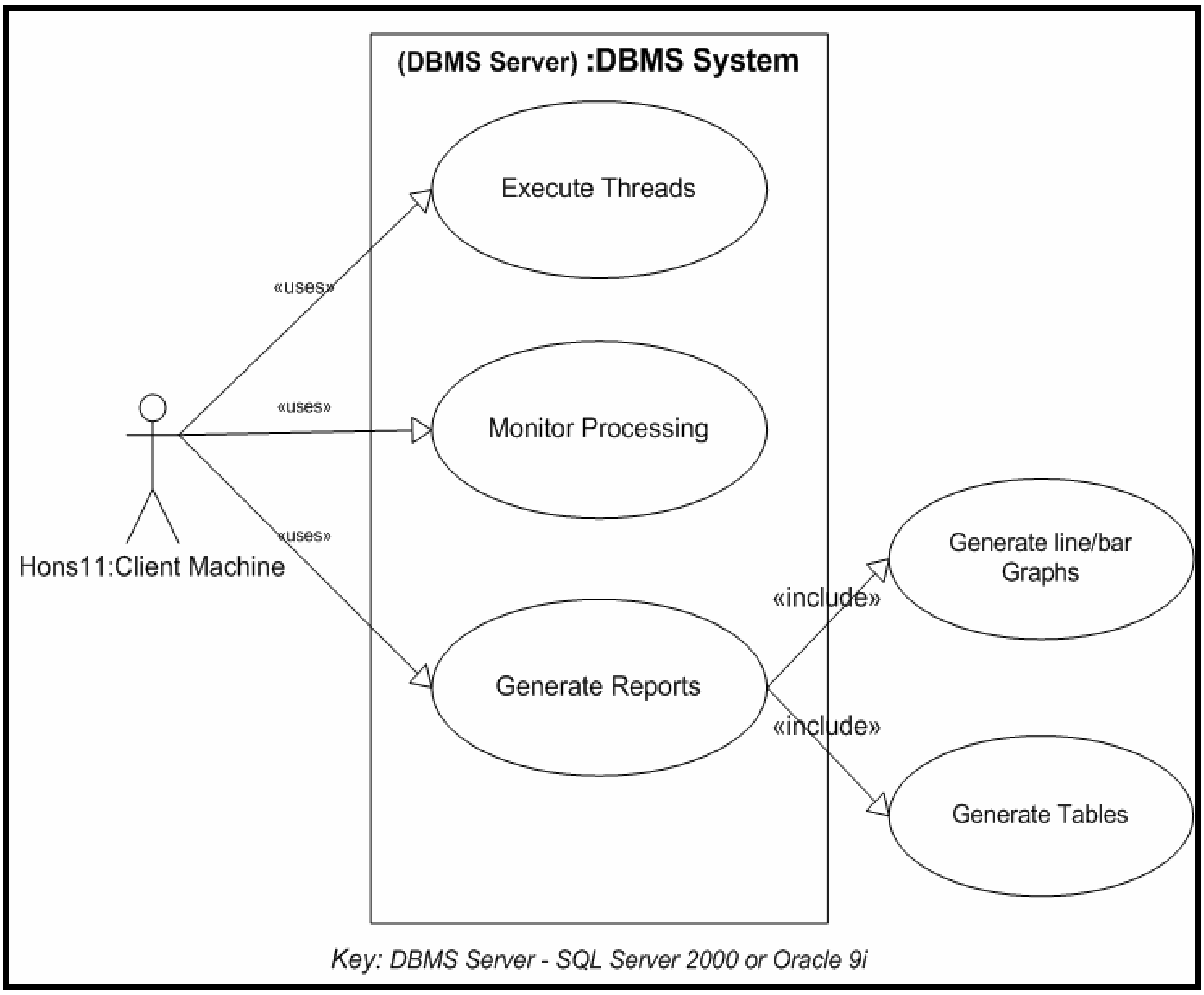


Figure 2-2 *UML diagram showing stress testing*

2.3.2.2.3 UML Description

Figure 2-2 shows a UML model of the stress testing process. On the diagram there are two systems operating. There is the actor (called Hons11) which is the controller machine and the system being modelled, the actual server which could is either SQL Server 2000 or Oracle 9i. During stress testing, threads were initiated from the client machine while on the DBMS system thread execution and monitoring using the Profiler (SQL Server 2000) and Statspack (Oracle 9i) were the major processes. Another important sever process was report generation which involved drawing tables and graphs.

2.4 Data Capture and Presentation

2.4.1 Data capture

Data capture for both load and stress testing was mostly done by System Monitor since besides allowing the author to keep track of real-time server activity it also enabled him to capture and save data onto log files for later scrutiny. Third party software used for this process basically included Benchmark Factory, Quest Central and Spotlight. Other tools which provided statistics for SQL Server 2000 included the Query Analyzer and the Profiler. SQL Query Analyzer has a powerful functionality for displaying execution plans used by the SQL Optimiser when executing the queries. Another of its strengthens including the Show Server Trace, Manage Statistics and Show Client Statistics services which I used to gather various statistical information for analysis. Oracle 9i provides a powerful Enterprise Manager which I extensively used in my project[5] .

2.2.2 Presentation and Analysis of results

A number of tools were used to present recorded data from performance tests. Microsoft Excel was used for drawing most graphs otherwise other graphs were taken from tools like Quest Spotlight. Graphs were sketched for:

- Throughput versus user load.
- Response Time versus user load.
- Resource Utilization versus user load.

Resource utilization comprised CPU, network I/O, disk I/O, and memory utilisation.

2.5 Chapter Summary

This chapter gives detailed information about the considerations and design of experiments used in this research work. The system described here included three machines, the client machine and two servers (SQL Server 2000 and Oracle 9i). It starts by the performance and scalability testing environment. Functional, performance and scalability tests considerations are also discussed. A UML Use Case diagram was used to explain the functionality of the system used to perform stress testing. Lastly, this chapter gives information about data capture and presentation issues. The next chapter gives the implementation details of the tests as well as the associated results.

[5] Refer to Appendix A-D for more information on the use of various tools used for this process.

Chapter 3

Functional, Performance and Scalability Tests and Results

Chapter Highlights

- Functional Tests
- Performance Tests
    - Load Testing
    - Stress Testing
- Scalability Tests
- Chapter summary

Chapter 3 : Functional, Performance and Scalability tests and Results

This chapter gives details of the implementation of the performance and scalability tests introduced in the design phase of chapter 2. It starts by the functional testing process and then the performance tests. The final section looks at the implementation of the scalability tests. Results for each section are also stated.

3.1 Functional Tests

Functional testing was performed using System Monitor and Quest Spotlight. The following tables show base processes and their related resource utilisation statistical information.

3.1.1 Client/Controller machine Baseline

Figure 3-1 is the baseline for the client machine. It shows resource usage percentages for particular objects (processor, memory, disk I/O and the network interface) for the client machine before any tests were taken. Also stated are system and user processes which were running on the machine prior to testing .Only system processes with high resource utilisation were included in this baseline.

Controller Machine Resources		
Type of Resource	**Metrics**	**Ave. Values measured**
Processor	% Processor Time	46.032
Memory	% Memory Utilisation	0.100
Physical Disk	% Disk Time	0.767
Network Interface	Bytes Received/sec	760.474
	Bytes Sent/sec	61.816
	Current Bandwidth	100000000

Table 3-1 *Table shows client machine resource consumption statistics as measured using System Monitor*

Base processes which contributed to the resource usage figures displayed in table 3-1:

Sqlserver.exe –Memory Usage 10,195k
System Idle Process-Memory Usage 56k
mdm.exe –Memory Usage 2.864k
SavRoam.exe –Memory Usage 5.548k
svchost.exe -Memory Usage 27.644k
savRoam.exe –Memory Usage 3.548k

Issa.exe- Memory Usage 10.325k
services.exe-Memory Usage 7.064k
winlogon.exe-Memory Usage 4.0651k
smss.exe –Memory Usage 408k
cidaemon.exe –Memory Usage 223k

3.1.2 SQL Server 2000 Baseline

Table 3-2 is the baseline list of objects taken for SQL Server 2000. This baseline information was used for identifying and separating server base processes from processes resulting from testing.

Object	Counter	Average Measured Values (01:00 to 03:00 , Sunday 16/10/05)
Memory	% Memory Usage	10.2
Processor	% Processor Time	4.481
Physical Disk	Avg. Disk Bytes/Transfer	16389.000
Logical Disk	% Disk Time	6.123
Network Interface	Bytes Received/sec	0.000

Table 3-2 *Table shows SQL Server 2000 resource consumption statistics while at rest as measured using System Monitor*

Baseline processes include:

System Idle Process-Memory Usage 56k	java.exe –Memory Usage 20.965k
services.exe -Memory Usage 7.064k	logons –Memory Usage 1.254k
mdm.exe –Memory Usage 2.864k	winlogon.exe-Memory Usage 4.0651k
SavRoam.exe –Memory Usage 5.548k	smss.exe –Memory Usage 408k
svchost.exe -Memory Usage 3.644k	cidaemon.exe –Memory Usage 223k
savRoam.exe –Memory Usage 3.548k	
ssqService.exe –Memory Usage 10.576k	
quest_launcher.exe –Memory Usage 1.906k	

3.1.3 Oracle 9i Baseline

Table 3-3 below is the Oracle 9i baseline. Like the previous regimen it gives Oracle 9i base processes and their related resource consumption values which were subtracted from the performance test results to get only performance testing results.

Object	Counter	Average Measured Values (01:00 to 03:00 , Sunday 16/10/05)
Memory	% Memory Usage	8.10
Processor	% Processor Time	13.001
Physical Disk	Avg. Disk Bytes/Transfer	7394.000
Logical Disk	% Disk Time	1.913
Network Interface	Bytes Received/sec	1499.177

Table 3-3 *Table shows Oracle 9i resource consumption statistics as measured by System Monitor*

Oracle 9i system processes included:

Sqlserver.exe –Memory Usage 21,023k	Issa.exe- Memory Usage 10.325k
System Idle Process-Memory Usage 99k	services.exe-Memory Usage 21.10k
System –Memory Usage 232k	winlogon.exe-Memory Usage 4.8281k
SavRoam.exe –Memory Usage 5.854k	smss.exe –Memory Usage 408k

svchost.exe -Memory Usage 27.644k	cidaemon.exe –Memory Usage 54k
dbtools.exe –Memory Usage 5.296k	agntsvc.exe – Memory Usage 2.456k
oracle.exe –Memory Usage 139.183k	ONRSD.EXE –Memory Usage 5.085k

3.1.3 Baselines and Performance Test results comparison

Table 3-1 through 3-3 above presented baselines for the three machines used in the tests as depicted by figure 2-1. Table 3-4 below shows a summary of some performance load testing results which illustrate how resource consumption changed for each machine with increasing load (number of users) as compared to the respective baselines.

SQL Server 2000 Resource Utilisation										
Users	80	160	240	320	400	480	560	640	720	800
% Processor Time Used	5.7	8	20	43	71	73	79.1	85	88	91
% Memory Usage	22.1	18.7	18.1	21.2	23.2	27.04	33.9	41.9	45.3	52
% I/O Usage (Avg. Disk Bytes/Transfer)	8.1	12.3	15.1	17.4	15.6	20.9	25.4	28.3	29	33.2
Oracle 9i Resource Utilisation										
Users	80	160	240	320	400	480	560	640	720	800
% Processor Time Used	11	15	18	43	47	49	55	60	71	76
% Memory Usage	18	19.6	19.8	18	19.3	19.7	19.8	19.06	19.35	19.7
% I/O Usage (Avg. Disk Bytes/Transfer)	9	11	14.9	16	17.3	21.5	25.6	27.5	32.3	28.3

Table 3-4 *Summary of SQL Server 2000 and Oracle 9i Load Testing results* (*Scenario 1)*

A comparison of the baselines with the above figures one can easily see the pattern in resource utilisation for each database making the result analysis process reliable and potentially simple. For instance one can analyses the % Processor time for SQL Server 2000 by noting that initially in the baseline the server used 4.481 % of the CPU cycles but the moment it was loaded with 80 users CPU percentage usage escalated to 5,7 % thus showing the effect of the load.

3.2 Performance Tests

This section presents performance tests which include load and stress testing. Each process was performed for three distinct areas that include response time, throughput rates and resource utilisation. For load testing three scenarios were used to investigate the three aspects mentioned above while for stress testing one scenario was used to evaluate all three areas according to the specifications in chapter 2 for loading and stressing the system.

3.2.1 Load Testing

Chapter 2 identified the following load testing steps:

i. Identification of key scenarios
ii. Workload modelling
iii. Identification of metrics
iv. Creation of test cases
v. Load and data simulation
vi. Data collection and analysis

3.2.1.1 Scenario 1: Testing Server Response Time

Key Scenario Used:

Scenario 1 was used to evaluate how the DBMS behave with respect to response time as the user load increases. Response time was measured as the total time the server took to execute and return complete results of a query weighted by the frequency of each sequence in the mix[6].

The scenario used included updating/editing the database tables according to the workload model specified below. Basically the query included operations such as inserting and deleting data from the database tables.

Definition of the workload model:

To emulate real production traffic a small percentage of concurrent users was used in the testing process. A minimum of 800 users were used for this test of which 80% of them represented simultaneous users while the remaining 20 % were concurrent users. Moreover 50% of the users executed an insertion query while the other 50% performed deletions to produce a 32% percentage mix.

Identification of metrics:

A combination of metrics were used for this process which include among others Transactions/sec, % Processor Time/_Total, Page Reads/sec Available Bytes, Lock Requests/sec, Average Wait Time (ms) and Full Scans/sec [APPENDIX E gives the a full list of metric definitions].

[6] Scenario mix is basically the ratio of concurrent users to simultaneous users in the workload profile weighted with the available actions among which the load is distributed, for instance Insertion and Deletion user actions. Percentage mix calculation for Scenario 1: 20/100 *80/100 * 50/100 * 50/100 = 32%

Test Case Description:

- 800 users were used for the test.
- A random Think Time[7] in the script configuration was specified to be between 1 and 10 seconds.
- Keying Time also ranged from 1 to 10seconds. This time interval creates a delay before a transaction executes, simulating activities such as data entry that a user performs before executing the transaction.
- Inter-arrival Time was also set to be between 1 and 10 seconds .Inter-arrival Time is the rate at which transactions are arriving to a server.
- Tests run for 2 hours each.

Load and Data simulation and ***Data collection and analysis:***

Load, data simulation and data collection and analysis steps were combined to one phase since they are performed simultaneously. Table 3-5 below shows response time figures captured using a combination of System Monitor, Quest Central and Spotlight.

Response Time in seconds		
Users	**Oracle9i**	**SQL Server 2000**
40	18.672	18.272
80	16.575	16.784
120	16.788	16.965
160	16.909	16.937
200	16.838	17.436
240	16.986	18.914
280	16.937	21.583
320	16.941	22.283
360	17.129	25.417
400	17.356	26.798
440	17.197	30.857
480	17.515	30.796
520	17.717	36.684
560	17.973	41.589
600	17.937	46.567
640	17.481	52.601
680	18.809	57.681
720	19.447	62.618
760	19.44	63.251
800	19.643	66.499

Table 3-5 *Response Time in seconds for 800 users*

[7] Random think time is the time spent by the user between two consecutive requests. This is the time when a user thinks of what to search for in the database

Figure 3-1 below shows a graph plotted using response time results in Table 3-5 above. The graph shows that at lower user load from 1 to about 200 users, response time for both servers was relatively the same. As the user numbers increased, exciting more pressure on the DBMS the response time started increasing steadily for SQL server 2000 while Oracle 9i remained with a relatively flat curve with about 16 – 17 seconds. At extreme load response time for SQL server 2000 increased sharply to a maximum of 67 seconds for 800 users. Oracle 9i response time remained fairly low with a maximum of 19 seconds for 800 users. Generally, response time results show that Oracle 9i is much faster than SQL server 2000 at extreme load while at low load both DBMS perform in a similar way.

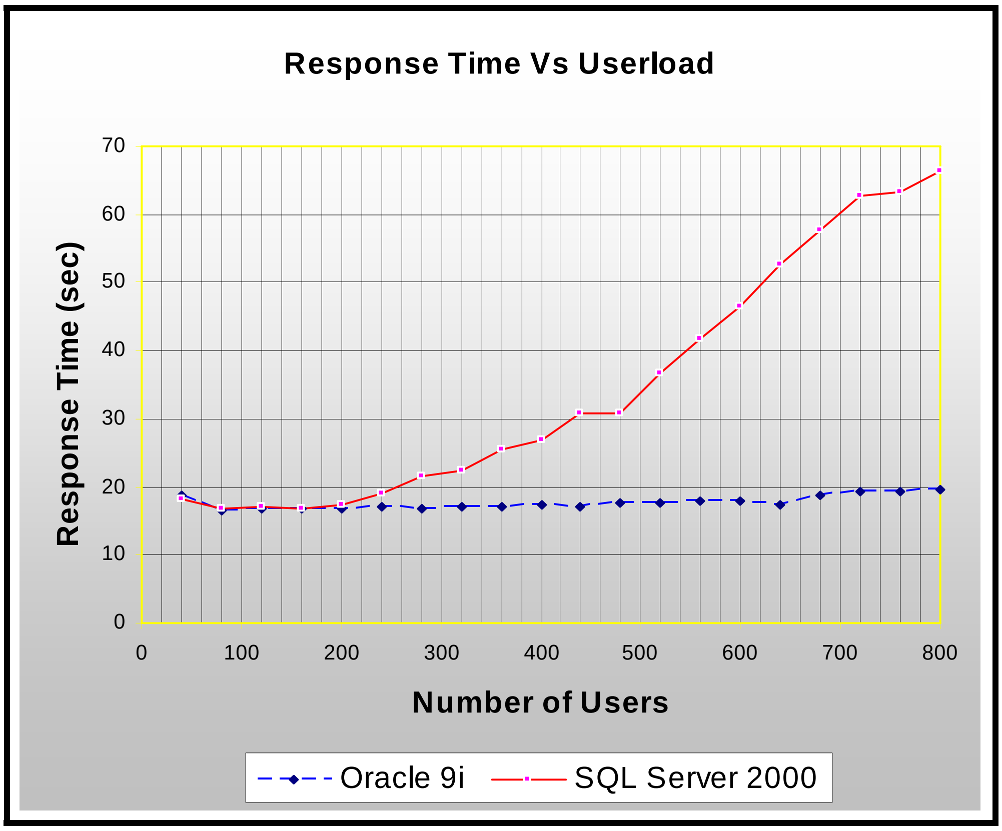


Figure 3-1 *The Graph shows Response Time plotted against the number of users*

3.2.1.2 Scenario 2: Testing Server Throughput Rate

Scenario Description:

The scenario used for this investigation was simply searching the database using a query which returned a large set of data items based on the specified criterion. The servers had to make full scans of all database tables before returning a full list of the records in order.

Definition of the workload model, Identification of metrics and ***Test Case Definition*** steps were omitted here because they are the same with those of the first scenario.

Load and Data simulation and ***Data collection and analysis:***

Table 3-6 below shows throughput rates measured in request per second for Oracle 9i and SQL Server 2000.

Throughput Rates (records per second)		
Users	**Oracle9i**	**SQL Server 2000**
40	30	40.7
80	4.021	42.5
120	45.647	44.347
160	92.812	85.7
200	98.717	96.724
240	141.706	127.7
280	147.4	143.076
320	193.7	175.2
360	196.653	178.406
400	230	202.4
440	244.341	196.676
480	273.9	217.5
520	290.318	205.9
560	317.6	212.7
600	336.418	206.153
640	354.4	219.9
680	381.671	202.335
720	395.7	219.1
760	423.929	203.318
800	440.201	211.8

Table 3-6 *Throughput Rates for 800 users in Records per second*

Figure 3-2 below is a graph showing throughput rates in records per second for both DBMS engines. At low to medium load both systems show rapid increases in the number of records returned. Initially between 1 and 120 users Oracle 9i returns fewer records than SQL Server 2000 (with 40 ,80 and 120 users, Oracle 9i returned 30, 4, and 45 records per second as compared to 40,42,44 records per second for SQL Server 2000). There are many explanations for this phenomenon but one possibility is that Oracle 9i does more initialisations of other involved components than SQL Server 2000. This means therefore that while Oracle 9i was busy initialising its components SQL Server 2000 was ready to service requests. However after that phase Oracle 9i throughput rate increased higher than that of SQL Server 2000 for the period between low to medium load (200 and 400 users). The graph for SQL Server 2000 flattened from 400 to 800 users with about 196 to 211 records per second while Oracle 9i graph steepened to a maximum of 440 records per second for 800 users.

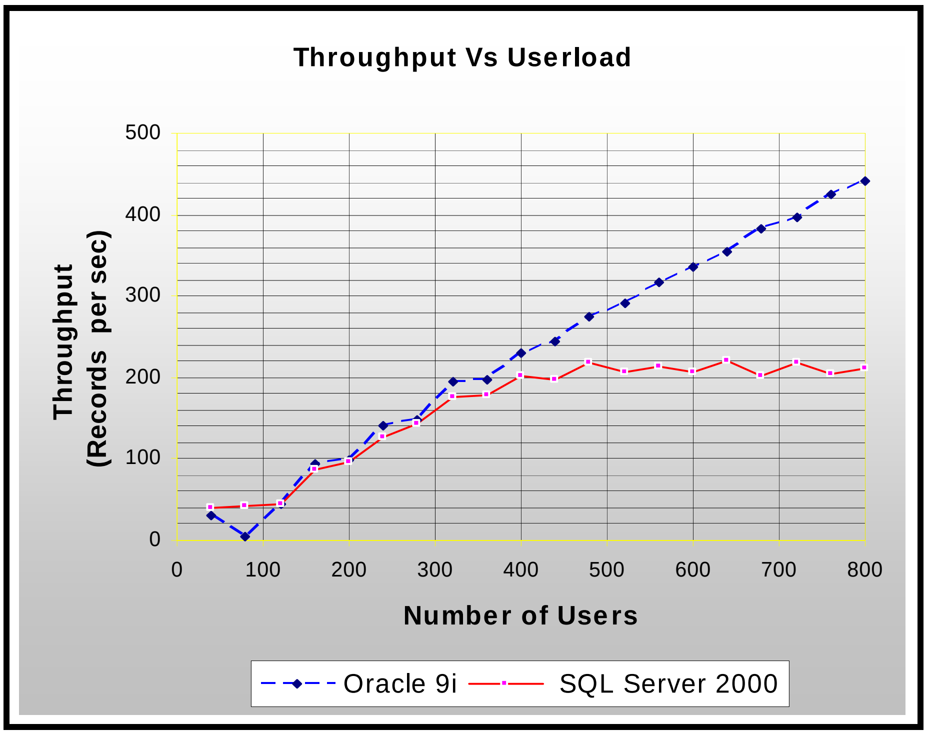


Figure 3-2 *The Graph shows Throughput Rates plotted against the number of users*

3.2.1.3 Scenario 3: Testing Server Resource Utilization

Scenario Description:

Resource utilisation was tested using the update (insert and delete) query similar to the one used for Scenario 1. This is because updating database tables is resource intensive thus makes a good measure for resource utilisation.

Definition of the workload model, Identification of metrics and ***Test Case Definition*** steps were omitted here because they are the same with those of the first scenario.

Load and Data simulation and ***Data collection and analysis:***

Table 3-7 and 3-8 below shows the average resource utilization percentages. Table 3-7 shows resource usage percentages for Oracle 9i while Table 3-8 shows those for SQL Sever 2000.

Oracle 9i Resource Utilisation										
Users	80	160	240	320	400	480	560	640	720	800
% Processor Time Used	11	15	18	43	47	49	55	60	71	76
% Memory Usage	18	19.6	19.8	18	19.3	19.7	19.8	19.06	19.35	19.7
% I/O Usage (Avg. Disk Bytes/Transfer)	9	11	14.9	16	17.3	21.5	25.6	27.5	32.3	28.3

Table 3-7 *Table shows Oracle 9i average percentage resource utilisation*

SQL Server 2000 Resource Utilisation										
Users	80	160	240	320	400	480	560	640	720	800
% Processor Time Used	5.7	8	20	43	71	73	79.1	85	88	91
% Memory Usage	22.1	18.7	18.1	21.2	23.2	27.04	33.9	41.9	45.3	52
% I/O Usage (Avg. Disk Bytes/Transfer)	8.1	12.3	15.1	17.4	15.6	20.9	25.4	28.3	29	33.2

Table 3-8 *Table shows SQL Server 2000 average percentage resource utilisation*

Figure 3-3 and 3-4 graphs respectively show processor and memory utilisation comparisons between the two servers.

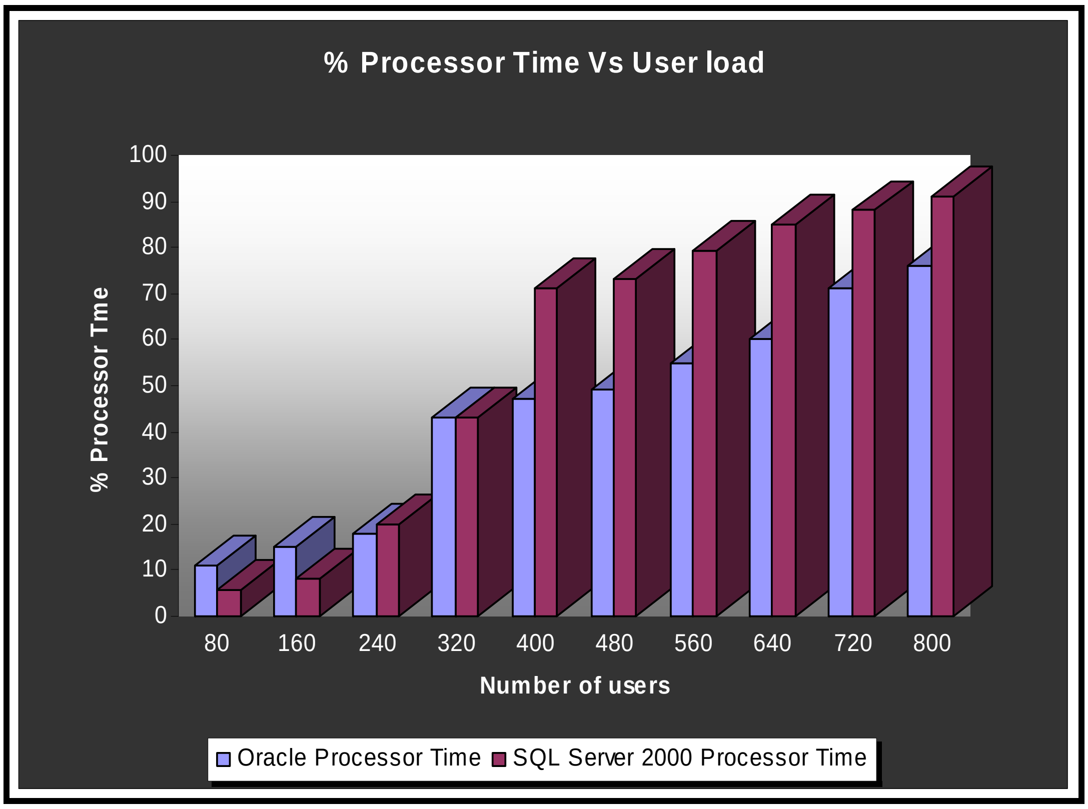


Figure 3-3 *The Graph shows % Processor Time plotted against the number of users*

Figure 3-3 illustrates that at low user loads (1 to 240 users) both system seem to optimally use the server resources. From 400 to 800 users resource consumption spiral simultaneously for both servers although SQL Server 2000 was more processor intensive measuring with 91% percentage Processor Time Used for 800 users as compared to 76% for Oracle 9i. Figure 3-4 below gives the memory usage graph for SQL Server 2000 and Oracle 9i.

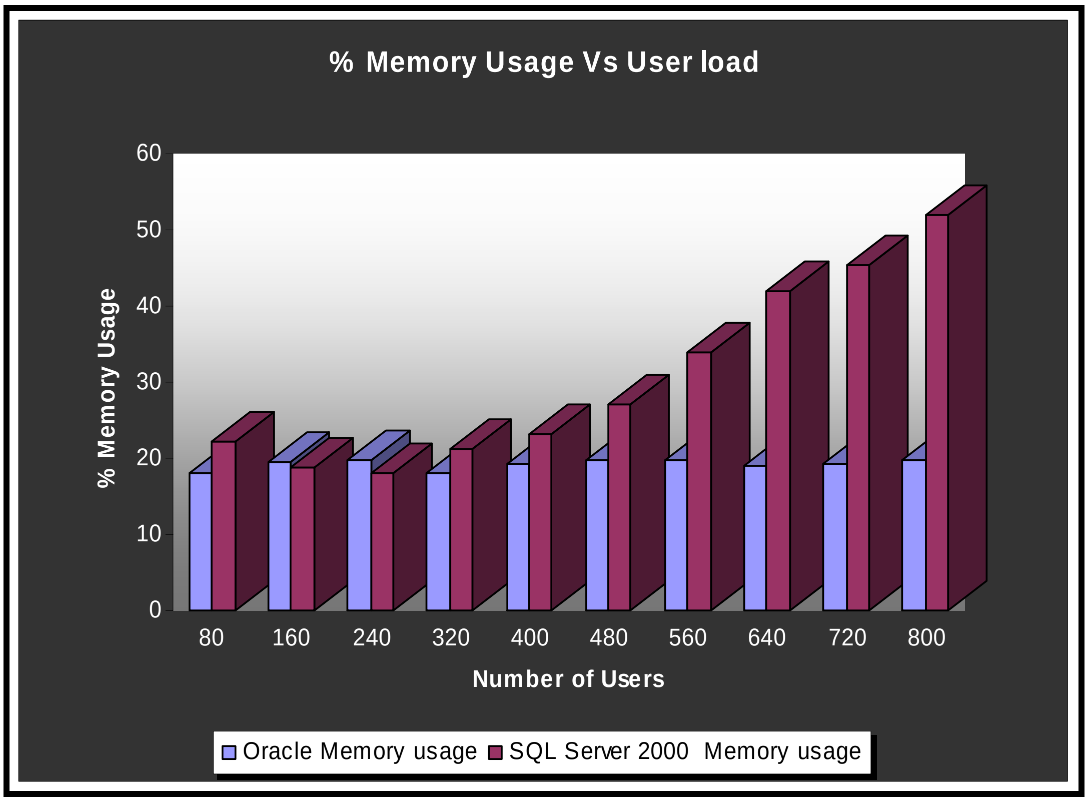


Figure 3-4 *The Graph shows % Memory Usage plotted against the number of users*

Figure 3-4 is the memory usage graph which shows that at extreme load SQL Server 2000 not only used the most CPU cycles [Figure 3-3] but also consumed more memory than Oracle 9i.

Figure 3-5 below compares I/O Subsystem percentages for Oracle 9i and SQL Server 2000. Disk I/O Subsystem was measured in Average Disk Bytes/Transfer. % disk time for both servers increased steadily at low to medium load. At extreme load SQL Server 2000 suddenly produced a higher % disk time (% 33.2 as compared to 28.3 % for Oracle)

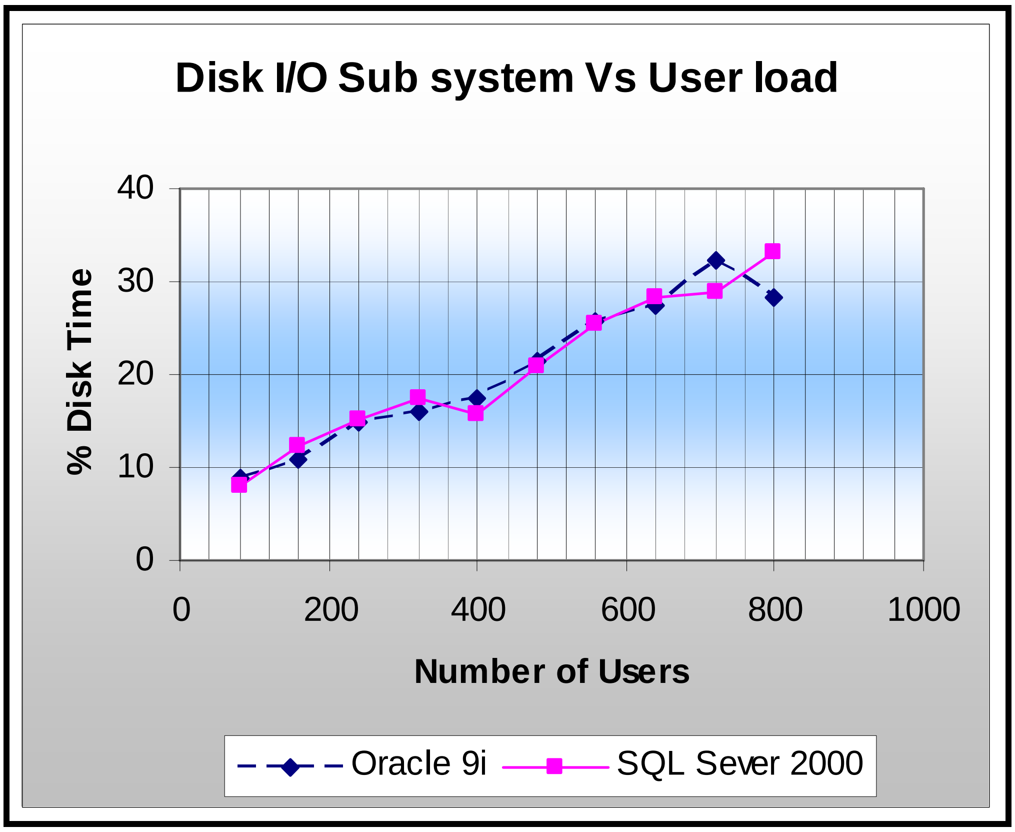


Figure 3-5 The Graph shows % I/O Subsystem Usage plotted against the number of users

The following Table 3-9 shows the average resource utilisation results calculated by halving CPU and memory usage percentages. Disk I/O Subsystem was measured and plotted as a separate entity [Figure 3-5].

AVERAGE RESOURCE UTILIZATION (%) =(CPU + Memory Utilization)/2		
Users	SQL Sever 2000	Oracle 9i
80	13.9	14.5
160	13.35	17.3
240	19.1	18.5
320	32.1	30.5
400	47.25	33.2
480	50	34.35
560	56.6	37.4
640	63.45	39.5
720	66.6	45.15
800	71.5	47.85

Table 3-9 *Table shows average resource utilisation results for Oracle 9i and SQL Server 2000*

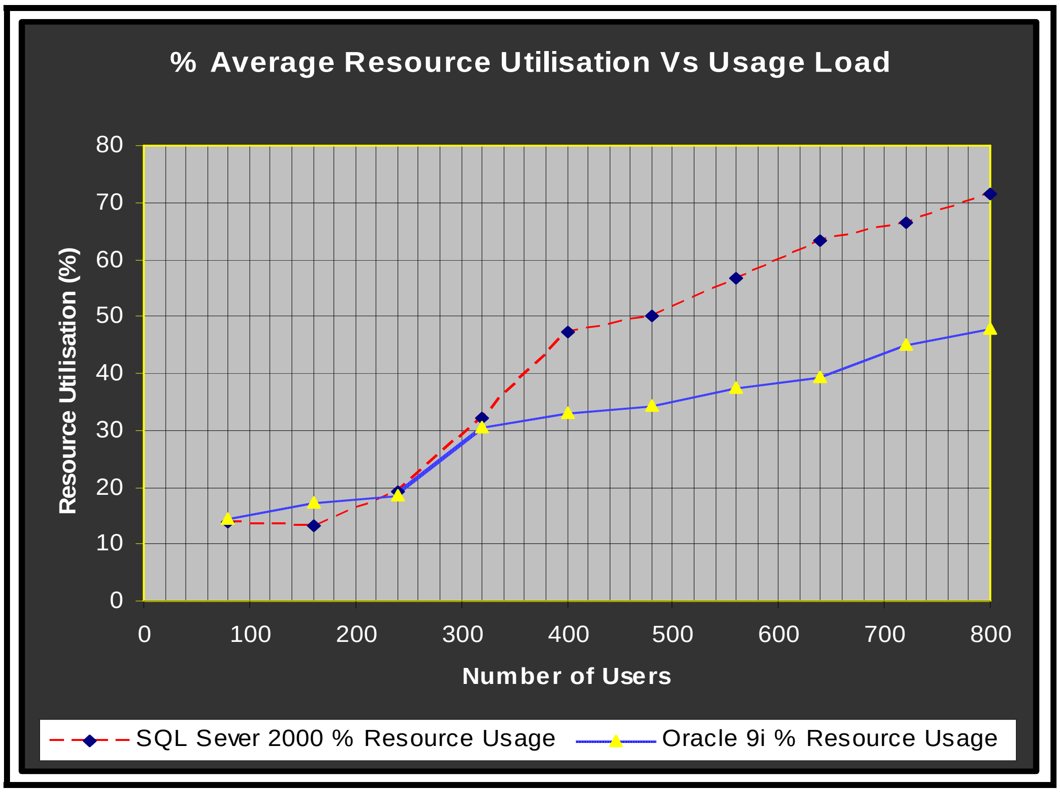
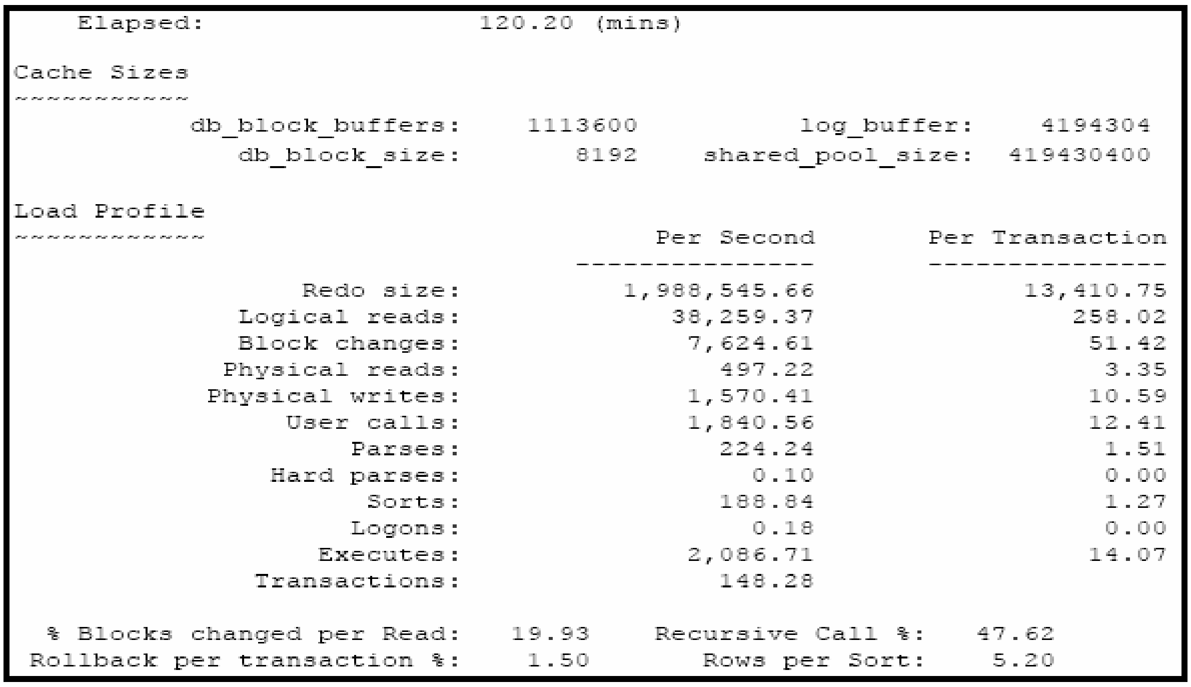


Figure 3-6 *The Graph shows Average Resource Utilisation plotted against the number of users*

Figure 3-6 above shows the combined average percentages of processor and memory utilization which were used to reflect average DBMS performance. These results show how resource efficient a DBMS is. The graph shows that at low load Oracle 9i generally consumed more memory and processor cycles (14 -17 %) than SQL server 2000 (13-14 %) but as load increased the reverse was true. At extreme load SQL server 2000 was more resource intensive than Oracle 9i consuming 75.5 % of its resources as compared to 47.85 for Oracle 9i.

One of the tools used by Oracle 9i for monitoring and data collection when testing resource utilisation was Statspack. An equivalent tool, the Profiler, was used for the same process by SQL Server 2000. Figure 3-7 and 3-8 below shows screenshots which illustrate how one can connect to Statspack and prepare for data collection and monitoring.

Figure 3-7 specifically shows the different types of information to expect when using Statspack. Importantly the figure shows the required and configured metrics and the time consumed by the test query per second/transaction. Metrics here were configured in line with the project. Details of how to use this tool are given in APPENDIX C.



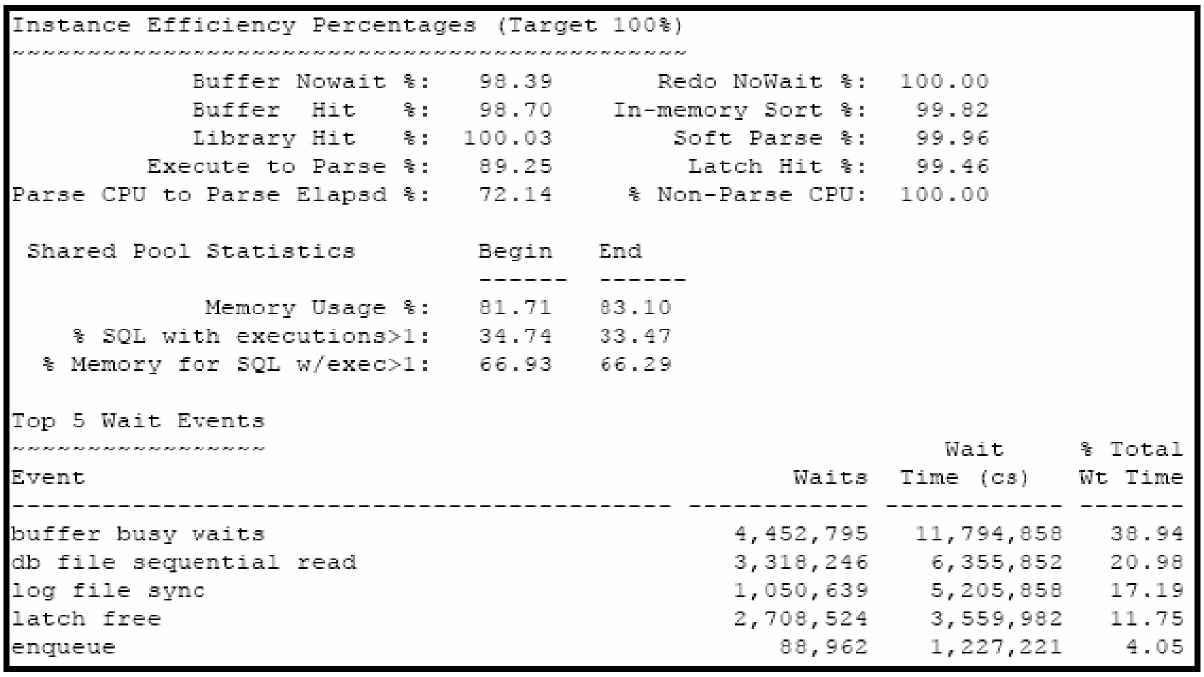
```
    Elapsed:                  120.20 (mins)

Cache Sizes
~~~~~~~~~~~
           db_block_buffers:    1113600        log_buffer:    4194304
              db_block_size:       8192   shared_pool_size:  419430400

Load Profile
~~~~~~~~~~~~                        Per Second       Per Transaction
                                ---------------      ---------------
                  Redo size:       1,988,545.66            13,410.75
              Logical reads:          38,259.37               258.02
              Block changes:           7,624.61                51.42
             Physical reads:             497.22                 3.35
            Physical writes:           1,570.41                10.59
                 User calls:           1,840.56                12.41
                     Parses:             224.24                 1.51
                Hard parses:               0.10                 0.00
                      Sorts:             188.84                 1.27
                     Logons:               0.18                 0.00
                   Executes:           2,086.71                14.07
               Transactions:             148.28

  % Blocks changed per Read:   19.93    Recursive Call %:   47.62
 Rollback per transaction %:    1.50       Rows per Sort:    5.20
```

Figure 3-7 *The Graph shows Statspack screenshot 1*



```
Instance Efficiency Percentages (Target 100%)
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
            Buffer Nowait %:   98.39       Redo NoWait %:  100.00
            Buffer  Hit   %:   98.70    In-memory Sort %:   99.82
            Library Hit   %:  100.03        Soft Parse %:   99.96
         Execute to Parse %:   89.25         Latch Hit %:   99.46
Parse CPU to Parse Elapsd %:   72.14     % Non-Parse CPU:  100.00

 Shared Pool Statistics        Begin   End
                               ------  ------
             Memory Usage %:   81.71   83.10
    % SQL with executions>1:   34.74   33.47
  % Memory for SQL w/exec>1:   66.93   66.29

Top 5 Wait Events
~~~~~~~~~~~~~~~~~                                             Wait     % Total
Event                                               Waits  Time (cs)   Wt Time
-------------------------------------------- ------------ ------------ -------
buffer busy waits                               4,452,795   11,794,858   38.94
db file sequential read                         3,318,246    6,355,852   20.98
log file sync                                   1,050,639    5,205,858   17.19
latch free                                      2,708,524    3,559,982   11.75
enqueue                                            88,962    1,227,221    4.05
```

Figure 3-8 *The Graph shows Statspack screenshot 2*

3.2.2 Stress Testing

Inputs to Stress Testing included the identified critical scenario, workload models and the peak load capacities obtained from the load testing process. This load was used as a guideline for the definition of the workload profile as the starting point in stressing the servers.

Steps followed for stressing were explained in the preceding chapter and are basically the same to those used for load testing:

Identification of key scenarios:

Guidelines mentioned in chapter 2 on the identification of the stress testing scenario were used to choose two possible scenarios for testing.

Possible Scenario 1

> Searching the database (Use a search query that returns a lager table and thus scans all tables) .This query had potential of impacting overall server performance with increased resource utilization (memory usage, CPU utilization and I/O disk subsystem).

Possible Scenario 2

> Use an Update query that specifies large input data.
>
> This scenario had potential of exhibiting intensive locking, synchronization, and I/O disk-intensive usage thus making a good scenario for stress testing.

Of the two scenarios identified above, the first one was chosen for this evaluation. Analysing resource utilization was more likely easier and feasible than evaluating locking, memory leaks and synchronization issues involved in the second scenario. However it would be interesting to note how the results would be like if scenario two was used.

Identification of workload and ***Identification of metrics*** were the same as in the previous section (load testing) thus the two steps were omitted here.

Create test cases:

- 800 concurrent users.
- A random Think Time of 0 seconds was used meaning all users fired requests at the same time.
- The tests run for 2 hour.

Load, Data simulation, Data collection and analysis:

Quest Central for both SQL Server 2000 and Oracle 9i was used for load simulation. Various tools were used for data capture and representation. For instance statistical information was obtained from SQL Server Query Analyzer (Execution plans) and the Profiler for SQL Server 2000 while staspack was used for Oracle 9i.

Tables 3-10 and 3-11 below represent resource utilisation results from stressing Oracle 9i and SQL Server 2000.

Oracle 9i Resource Utilisation										
Users	80	160	240	320	400	480	560	640	720	800
% Processor Time Used	18	20	26	32	42	58	66	70	76	88
% Memory Usage	36	40	50	63	78	81	82	80	88	91

Table 3-10 *Table shows average resource utilisation results for Oracle 9i*

SQL Server 2000 Resource Utilisation										
Users	80	160	240	320	400	480	560	640	720	800
% Processor Time Used	16	28	48	50	78	83	85	87	95	99
% Memory Usage	40	48	56	66	83	86	89	91	95	97

Table 3-11 *Table shows average resource utilisation results for SQL Server 2000*

Figure 3-9 is a processor line graph drawn from the statistics illustrated in Table 3-10 and 3-11. When stress tested both servers showed increase resource utilisation [Figure 3-9 and 3-10 below] as compare to the load testing results shown above. This is partly so due to the concept of concurrent and simultaneous users. For load testing a variable random think / keying time of between 1 and 10 seconds was used in the scripts with a combination of 80% simultaneous users and 20% concurrent users to simulate real production traffic. This distributed the load and frequency of requests at the servers. Foe stress testing the Keying time was set to 0 seconds thus all users were concurrently accessing the database which increased resource demand on the servers leading to high processor and memory consumption.

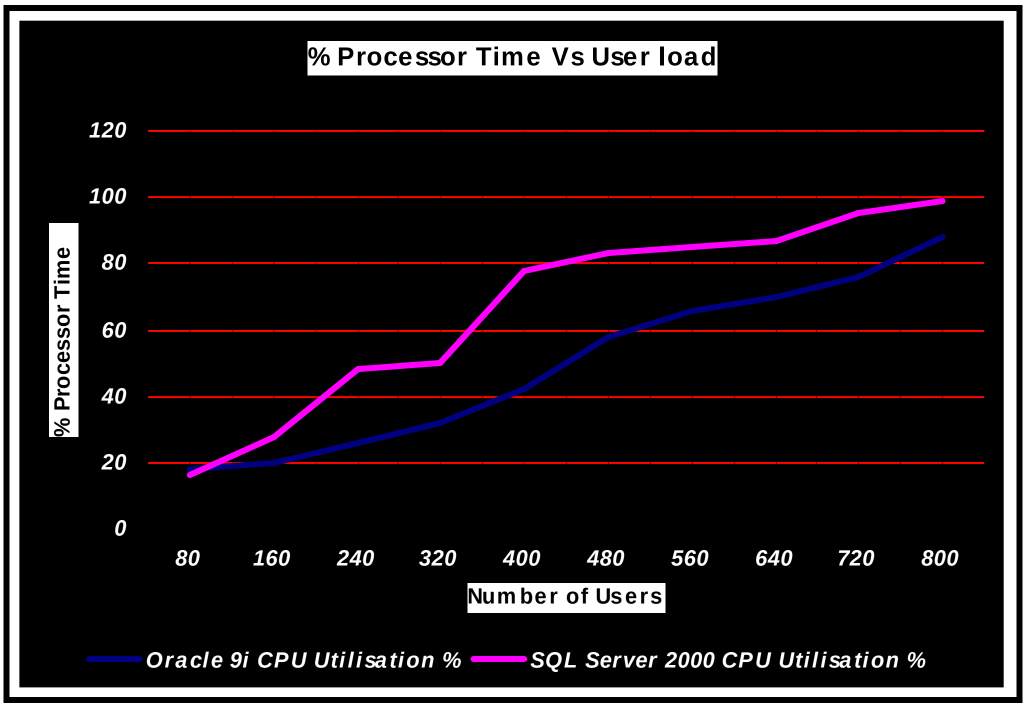


Figure 3-9 *The Graph shows % Processor Time Vs User load*

Figure 3-9 is a processor line graph drawn from the statistics illustrated in Table 3-10 and 3-11.

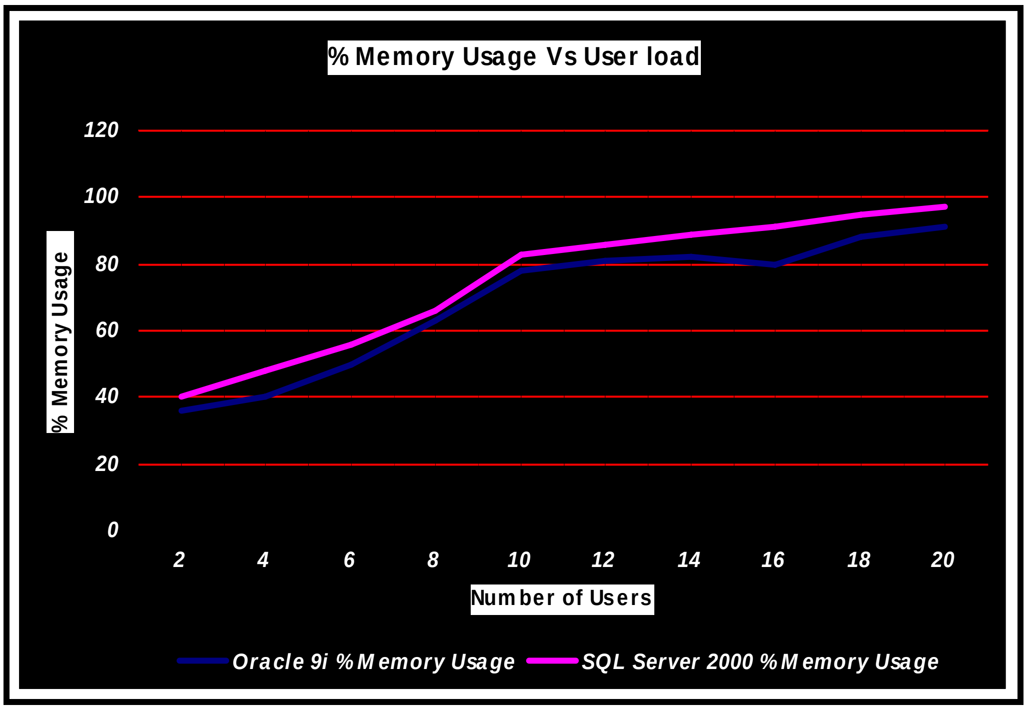


Figure 3-10 *The Graph shows % Memory Usage Vs User Load*

Table 3-12 and Figure 3-11 below show average resource utilization utilisation percentages and a line graph from the stress testing process.

AVERAGE RESOURCE UTILIZATION (%) =(CPU + Memory Utilization)/2		
Users	SQL Sever 2000	Oracle 9i
80	28	27
160	38	30
240	52	38
320	58	47.5
400	80.5	60
480	84.5	69.5
560	86	74
640	89	75
720	95	82
800	96.5	89.5

Table 3-12 *Table shows average resource utilisation results for Stress testing*

Due to lock, resource contention and synchronisation issues between concurrent readers and readers, SQL Server 2000 crashed with "Server busy" messages just after servicing about 880 simulating users. On the other hand Oracle 9i continued servicing request. This was probably due to its concurrency control mechanism (the Multi-Version Read Consistency) which enable many concurrent user to access the database. Threading was used for simulating the users during stress testing.

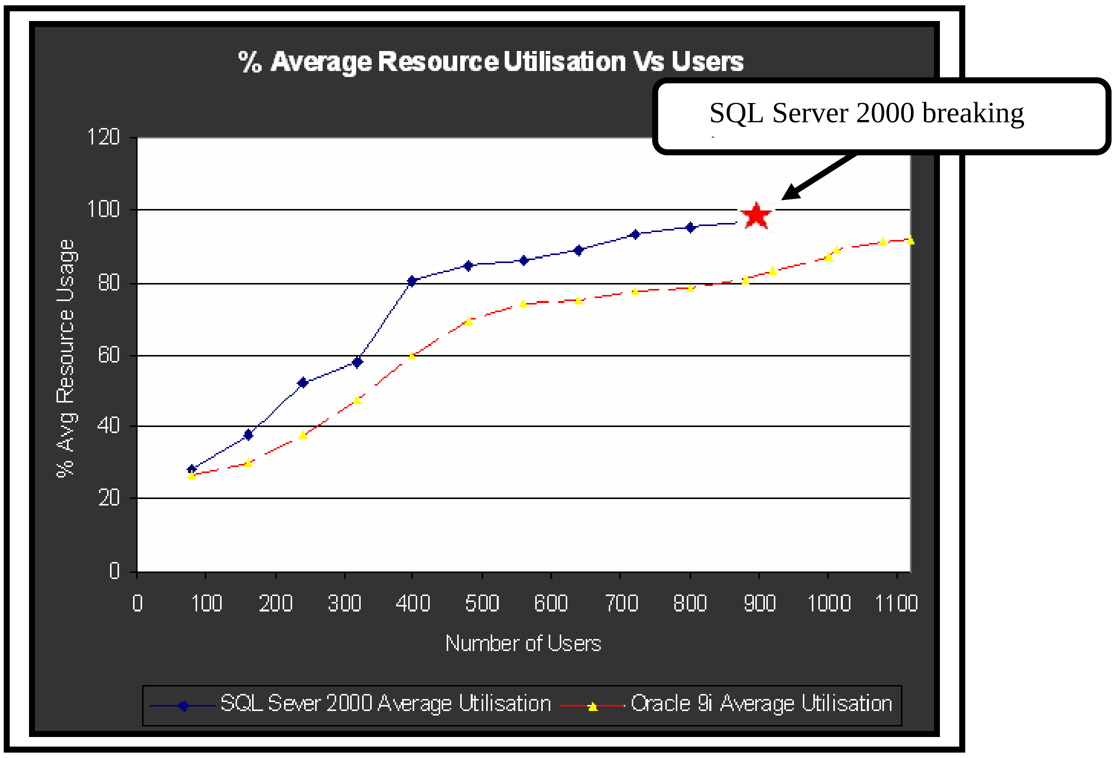


Figure 3-11 *The Graph shows average resource utilisation plotted against the number of users*

Figure 3-11 shows that at all levels of user load SQL Server 2000 responded with high resource utilisation than Oracle 9i. SQL Server 2000 failed with a maximum average resource utilisation of 96.5% which was more than 91.5 % for Oracle 9i with 1200 users.

3.3 Scalability Testing

Given the results above it is apparently clear that as the user numbers increased SQL Server 2000 resource consumption escalated while producing almost stagnant throughput rates [Figure 3-2]. Throughput rate results obtained from the second scenario in the Load testing process show that SQL Server 2000 reached threshold much faster than Oracle 9i. In addition when looking at the response time results one can see that in SQL Server 2000 as user populations increase response time increases to unacceptable levels. All this means that Oracle 9i scales better than SQL Server 2000 with increasing number of simultaneous and concurrent users. However a conclusive picture of scalability can be obtained by testing the DBMS systems under different software and hardware (RAM and number of processors) configurations in which case scalability will be measured as by noting the number of users the server is able to save as additional hardware in being added to the system.

3.4 Chapter Summary

Chapter 3 explains the implementation of the performance tests with the associated results. Aspects investigated include response time, throughput and resource utilization (processor, memory and I/O disk subsystem). The next chapter presents a further evaluation of the results obtained from this chapter and give other factors affecting DBMS performance and behaviour for modern database systems.

Chapter 4

Other Factors Affecting Performance and Scalability

Chapter Highlights

- Discussion/Analysis of results
- Database Engine Technical Comparison
  - The Concurrency model
  - Indexing
  - Partitioning capabilities
  - Parallel execution and clustering capabilities
- Chapter summary

Chapter 4 : Other Factors Affecting Performance and Scalability

This chapter analyses some factors affecting performance and scalability in modern DBMS systems. The architectural design of each DBMS system is use to explain the performance and scalability results obtained in chapter 3.

4.1 Analysis of results

Performance results unveiled in the preceding chapter could be analysed and evaluated further to ascertain which technical factors influenced the DBMS to react as they did during the tests. These factors either had a direct or indirect influence on performance and scalability as discussed in the following section. Aspects investigated in this project comprise the concurrency model, partitioning, database indexing and parallel execution capabilities. These factors were analysed in relation to their influence on response time, throughput rate and resource utilisation.

4.1.1 Concurrency model

The concurrency control in databases ensures that under multi-user environments data modifications done by one user do not negatively affect those done by another. SQL Server 2000 and Oracle 9i differ in the way they implement concurrency control mechanisms as explained in the following section.

4.1.1.1 Multi-Version Read Consistency

Below is a list of some possible problems most likely to be encountered in multi-user conditions that may have influenced response time and throughput rates for the first two scenarios in the load test process.

- *Dirty*, or uncommitted, are reads that happen when a transaction can read changes made to the database that have not yet been committed.
- *Non-repeatable reads* occur when a transaction re-reads data it has previously read and finds that another committed transaction has modified or deleted the data.

Oracle 9i implements a concurrency model where multi-version read consistency always provides consistent and accurate results. When an update occurs in a transaction , the original data values are recorded in the database undo records. Oracle 9i uses the current information in the undo records to construct a read-consistent view of table data, and to ensure that a version of the information, consistent at the beginning of the uncommitted transaction, can always be returned to any user. SQL Server 2000 on the other hand, have to choose a workaround to avoid concurrency problems, such as locking data to prevent it from changing while being read or preventing queries from reading changed but uncommitted information [MSDN library, 2005] ,[Microsoft SQL Server documentation, 2005].

SQL Server 2000 does not provide multi-version read consistency. Instead it requires transactions to either use shared locks for read operations, with various levels of isolation, or to accept dirty reads. Shared locks prevent data that is read from being changed by concurrent transactions [MSDN library, 2005]. It is clear therefore that this implementation restricts the ability of the system to properly service concurrent requests and users in environments involving a mix of reads and writes. This explains Figure 3-1 which represents response time plotted against the number of users where response time for SQL Server 2000 increased exponentially with increasing load, reads and writes had to be coordinated through shared locks which resulted in increased turnover time while Oracle 9i implementation of multi-version read consistency ensured that operations were freely executed thus reducing the response overhead.

To try and add variation to the tests for architectural analysis purposes the author had to build separate workload environments, some of which exhibited intensive read activities, such as selecting rows by a query which needed to scan the whole database. Regardless of my efforts, concurrency and accuracy undoubtedly affected throughput and scalability in SQL Server 2000 as shown in Figure 3-2. With 400 users simultaneously querying the database SQL Server 2000 throughput rates remained constant at about 200 records per second while for Oracle 9i which favours increased concurrent users due its sound concurrency model implementation throughput rates increased with the increasing number of users to any extreme of 400 records per second for 800 users. This was because Oracle 9i allowed writers and readers to operate cleanly in mixed workload environments without incurring any performance downtime.

4.1.1.2 Non-Escalating Row-Level Locking

Row-level locks offer the finest granularity of lock management, and thus, the highest degree of data concurrency. Row-level locking ensures that any user or operation updating a row in a table will only lock that row, leaving all other rows available for concurrent operations. Oracle 9i uses row-level locking as the default concurrency model and stores locking information within the actual rows themselves. By doing so, Oracle 9i can have as many row level locks as there are rows or index entries in the database, providing unlimited data concurrency [Balloni , 2000].

SQL Server 2000 on the other hand supports row-level locking as the default concurrency model. However, because it was not the default level of lock granularity in earlier versions of the database, the late addition of row-level locking was made possible only through the use of additional, separate pools of lock structures. As with any memory structure, these lock structures are limited in their size and thus limit the maximum number of locks that can be supported by the database. The maximum amount of memory devoted to locks is limited to 40% of the amount reserved for the database. According to information from the Microsoft documentation, SQL Server 2000 dynamically determines the appropriate level at which to place locks for each statement but the level at which locks are acquired does not depend on the memory available. For example a small table may have a table lock applied, while a larger table may have row locking applied. A consequence is that as more users access the application and transaction volume increases, SQL Server 2000 will escalate row level locks to table locks to conserve memory [Understanding Locking in SQL Server, 2005]. This in turn means that fewer users can access the data at the same time – users will have to wait which further supports the results obtained in the preceding chapter where Oracle 9i showed better response time and throughput rates.

4.1.2 Indexing capabilities

Indexes are database structures that are created to provide a faster path to database information. Using indexes can dramatically reduce disk I/O operations thus increasing the performance of a DBMS in data retrieval. Both Oracle and SQL Server 2000 support traditional B-Tree indexing schemes, which are ordered lists of key values, associated with the storage location of the table row that contains these values. Both also support index-organized tables, which are called clustered indexes by Microsoft experts. Index-organized tables provide fast access to table data for queries involving exact match and/or range search on the primary key because table rows are stored in the leaf nodes of the primary key index. However, Oracle9i also supports static bitmap indexes and bitmap join indexes, whose usage can provide huge performance benefits for typical load and query operations in data warehousing and multi-user environments.

4.1.2.1 Bitmap Indexes and Bitmap Join Indexes

A bitmap index uses a bitmap (or bit vector) for each key value instead of a list of the table rows' storage locations - ROWID[8] . Each bit in the bitmap corresponds to a row in the table. The bit is set when the table's row contains the key value. Bitmap representation can save a lot of space over lists of ROWID, especially for low cardinality data. Bitmap indexes lend themselves to fast Boolean operations for combining bitmaps from different index entries. Bitmap indexing efficiently merges indexes that correspond to several conditions in a WHERE clause. Rows that satisfy some, but not all, conditions are filtered out before the table itself is accessed. This improves response time dramatically.

With Oracle9i, it is also possible to create bitmap indexes on index-organized tables, thereby allowing index-organized tables to be used as fact tables in data warehousing environments.

A bitmap join index is a bitmap index for the join of two or more tables. A bitmap join index can be used to avoid actual joins of tables, or to greatly reduce the volume of data that must be joined, by performing restrictions in advance. Queries using bitmap join indexes can be sped up via bit-wise operations. Bitmap join indexes, which contain multiple dimension tables, can eliminate bitwise operations, which are necessary in the star transformation with bitmap indexes on single tables. Performance measurements performed under various types of star queries demonstrated tremendous response time improvements when queries used bitmap join indexes.

SQL Server 2000 does not support bitmap indexes and bitmap join indexes.

4.1.3 Partitioning

Partitioning allows large database structures (tables and indexes for instance) to be decomposed into smaller and more manageable pieces. Although it is primarily considered a feature for manageability and availability, partitioning also provides a number of performance benefits [Oracle9i partitioning, an Oracle white paper, May 2001].

4.1.3.1 Oracle9i Partitioning Options

Oracle 9i Database offers several table partitioning methods designed to handle different application scenarios:

- Range partitioning uses ranges of column values to map rows to partitions. Partitioning by range is particularly well suited for historical databases. Range partitioning is also the ideal partitioning method to support 'rolling window' operations in a data warehouse.

[8] Each ROWID represents the storage address of a row. A **physical** ROWID identifies a row in an ordinary table. A **logical** ROWID identifies a row in an index-organized table

- Hash partitioning uses a hash function on the partitioning columns to stripe data into partitions. Hash partitioning is an effective means of evenly distributing data.
- List partitioning allows users to have explicit control over how rows map to partitions. This is done by specifying a list of discrete values for the partitioning column in the description for each partition.
- In addition, Oracle supports range-hash and range-list composite partitioning.

Oracle 9i also provides three types of partitioned indexes:

- A local index is an index on a partitioned table that is partitioned using the exact same partition strategy as the underlying partitioned table. Each partition of a local index corresponds to one and only one partition of the underlying table.
- A global partitioned index is an index on a partitioned or non-partitioned table that is partitioned using a different partitioning-key from the table.
- A global non-partitioned index is essentially identical to an index on a non-partitioned table. The index structure is not partitioned. Oracle allows all possible combinations of partitioned and non-partitioned indexes and tables: a partitioned table can have partitioned and non-partitioned indexes and a non-partitioned table can have partitioned and non-partitioned indexes.

4.1.3.2 SQL Server 2000 Partitioning Options

SQL Server 2000 supports partitioned views. A partitioned view joins horizontally partitioned data from a set of member tables across one or more servers, making the data appear as if from one table. The data is partitioned between the member tables based on ranges of data values in one of the columns, called the partitioning column. The data ranges for each member table are defined in a CHECK constraint specified on the partitioning column. The view uses UNION ALL to combine selects of all the member tables into a single result set. SQL Server 2000 distinguishes between local and distributed partitioned views. In a local partitioned view, all participating tables and the view reside on the same instance of SQL Server. In a distributed partitioned view, at least one of the participating tables resides on a different (remote) server.

Due to its lack of real partitioning capabilities, SQL Server 2000 suffered from many important deficiencies which impacted performance and scalability.

4.1.3.3 Partition Pruning

Partition pruning enables operations to be performed only on those partitions containing the data that is needed. The query optimizer analyzes FROM and WHERE clauses in SQL statements to eliminate the unneeded partitions: partitions that do not contain any data required by the operation are eliminated from the search. This technique dramatically reduces the amount of data retrieved from disk and shortens the use of processing time, improving query performance and resource utilization. It is an essential performance feature for data warehousing environments. Oracle9i fully implements partition pruning, including with composite partitioned objects. In addition, partition pruning can be combined with index access (global or local). When the index and the table are partitioned on different columns, Oracle9i will eliminate unneeded index partitions, even when the partitions of the underlying table cannot be eliminated. SQL Server 2000 implements rudimentary partition pruning with local partitioned views, when all participating tables and the view reside on the same instance of SQL Server. If CHECK constraints are used on the partition columns when creating the member tables, the query optimizer will determine which member tables contains the rows and will limit the search to these tables. However, overall performance effectiveness is limited since this technique cannot be combined with global indexes and composite partitioning, which are not supported. Partition pruning with Distributed Partitioned Views is severely restricted and suffers from lack of scalability.

4.1.3.4 Partition-Wise Joins

With Oracle9i, partitioning can also improve the performance of multi-table joins, by using a technique known as partition-wise joins. Partition-wise joins can be applied when two tables are being joined together, and both of these tables are partitioned on the join key. Partition-wise joins break a large join into smaller joins that occur between each of the partitions, completing the overall join in less time. This offers significant performance benefits both for serial and parallel execution. SQL Server 2000 does not support partition-wise joins.

4.1.4 Parallel execution of operations

Parallel execution of SQL operations vastly improved the performance for operations involving large volumes of data such as full table scans that I implemented for scenario 1. It helped reduce response time for data-intensive operations on my large database. Oracle9i was able to execute INSERT, UPDATE and DELETE statements in parallel when accessing both partitioned and non-partitioned database objects. With SQL Server 2000, INSERT, UPDATE, and DELETE statements were only executed which further hindered throughput and increased response time leading to performance downtime [Jones, 1997, pg 181].

4.1.5 Clustering

One last factor the author would have wished to evaluate for these databases was their implementation of Clusters. Clusters are groups of independent servers, or nodes, connected via a private network (called a cluster interconnect), that work collaboratively as a single system. Clusters allow applications to scale beyond the limits imposed by single node systems when processing loads which exceed the capacity of large individual servers. They have a significant influence on performance and scalability.

The table below shows a summary of the technical comparison of the two DBMS systems with the respect to the above discussed factors that are most likely to have influenced performance and scalability during tests.

Factors influencing Performance and Scalability in modern DBMS systems		
	Oracle 9i	SQL Server 2000
Concurrency Model	Multi-version read Consistency	Shared read locks or dirty reads
	Non-Escalating row level locking	Locks escalate
	Minimal deadlocks under load	Deadlocks can be a serious problem under load
Indexing capabilities	B-Tree indexes	B-Tree indexes
	Index-organized Tables	Clustered Indexes
	Bitmap indexes	Not supported
	Bitmap Join Indexes	Not supported
Partitioning options	Range, hash, list and composite partitioning	Not supported
	Local and global indexes	Only local indexes with member tables
Parallel execution	Queries, INSERT, UPDATE, DELETE	Queries only

Table 4-1 *Summary of factors affecting Modern DBMS according to the Microsoft and Oracle 9i documentation*

4.1.6 Performance tuning aspects affecting performance and scalability

- SQL Server 2000 has no control over sorting (memory allocation) while Oracle 9i can fully control the sort area size and allows it to be set by the DBA.
- SQL Server 2000 has no control over SQL Caching (memory allocation) while this is otherwise in Oracle 9i.
- SQL Server 2000 has no control over storage/space management to prevent fragmentation. All pages (blocks) are always 8k and all extents are always 8 pages (64k). This means you have no way to specify larger extents to ensure contiguous space for large objects. On the other hand Oracle 9i, this is fully configurable.
- And lastly SQL Server 2000 has no Log miner facility. Oracle 9i supply a Log Miner which enables inspection of archived redo logs. This comes free with the database. But in the case of SQL Server, external products from other companies have to be purchased to do this important DBA task [At lease 10 difference between oracle 9i and SQL server 2000.htm].

All these factors influenced the performance and scalability of the databases during tests.

4.2 Chapter Summary

This chapter discussed factors most likely to have caused the DBMS to behave the way they did during the testing process. Investigated factors include the concurrency model (multi-version consistency and lock escalating capabilities), indexing, and partitioning and parallel execution capabilities. Also briefly discussed is the clustering capability. The next chapter is the last of this project and therefore gives a conclusion of issues in presented in the proceeding chapters.

Chapter 5

Conclusion and Possible Project Extensions

Chapter Highlights

- Conclusion
- Possible Project Extension
- References
- Appendix A-E

CHAPTER 5 : Conclusion and Possible Project Extensions

This presents the conclusion and final analysis of the results detailed in the previous chapters. Also of particular importance the possible project extensions are given below.

5.1 Conclusion

According to the results presented in the preceding chapters it is apparently clear that SQL Server 2000 is positioned between MS-ACCESS and ORACLE in terms of performance and scalability.
It makes a work group level solution with a relatively small number of simultaneous and concurrent users and for small amounts of data. On the other hand Oracle 9i is much more sophisticated with more complex functionalities for both Online Transaction Processing and Data Warehousing applications. Although the performance and scalability of a DBMS is highly dependent on to experience of the Database Administrator to fine tune his database factors presented in chapter 4, Oracle 9i is a true enterprise solution in multi-user environments.

5.1.1 Response Time Evaluation

Response time results presented in chapter 3 for Scenario 1 showed that both server at low load produce reasonable turnaround times. However because of the architectural design factors discussed in chapter 4, especially the implementation of the concurrency model, SQL Server 2000 response time increase far much faster than that of Oracle 9i. It is apparent therefore that Oracle has a better scaling mechanism with increasing populations. This conclusion further supports my conclusion that Oracle 9i is a true enterprise solution.

5.1.2 Throughput Rate Evaluation

Figure 3-2 represents the throughput rates obtained from Scenario 2 of the load testing process. The figure shows Oracle 9i throughput rate increasing unboundedly as compared to SQL Server 2000 whose graph flattened at about 200 requests per second. Regardless of the increasing numbers of user load the amount of the work done remained constant. This is a serious phenomenon for multi-user environments. This restricts SQL Server 2000 to smaller workgroup environments as compared to Oracle 9i which undoubtedly favours large corporation environments.

5.1.3 Resource Utilisation Evaluation

Scenario 3 of the load testing process and stress testing results show that SQL Server 2000 can perform better with an additional resource pools. The latest SQL Server 2000 federated and clustered database abilities enhance performance by adding more resources in terms of software and hardware. Despite these performance and scalability issues that SQL Server 2000 has it is a better database in terms of usability and maintainability as compared to Oracle 9i which quires skills and technical knowledge thus making it more expensive to maintain.

5.1.4 Other factors affecting performance and scalability

In chapter four factors most likely to have influenced performance and scalability tests were discussed. Form the architectural comparison the concurrency model, indexing capacities, partitioning and parallel execution factors were evaluated with respect to the performance and scalability results presented in chapter 3. These factors showed that SQL Server 2000 has an architecture that favours small working environments while Oracle 9i favours large corporate situations characterised by multi-user environments.

5.2 Possible Project Extensions

5.2.1 In-depth Scalability Testing

The first possible extension to this project is to perform a deeper evaluation of the DBMS not only through stress and load testing but using different system configurations to determine the effects of adding/subtracting hardware and /or software to distribute “work” among system components (clustering). Tests of this kind according to RPM Solutions Pty Ltd (2004) can be performed in a variety of configurations, with such variables as network speed, number and type of server/CPUs and memory closely monitored during scalability testing.

Oracle and Microsoft and other systems form different RDBMS vendors could be evaluated for performance and scalability using portioning, clustering and federated databases. This will require additional hardware and a lot of effort to accomplish.

5.2.2 Test different Operating Systems / latest DBMS systems

Similar performance and scalability experimental tests can be performed on a UNIX platform like Linux and results compared to these found from using a Windows NT platform for the same Database Management Systems and/or different systems. Moreover, experiments could be performed to investigate latest DBMS system from both Microsoft and Oracle Corporation which are Oracle10g and Microsoft Yukon.

5.2.3 DBMS technical/architectural comparison

Table 4-1 shows a table of factors that affect performance and scalability capabilities offered by modern DBMS systems. A possible project extension therefore might be to prove, using different code, to what extent these factors affect performance and scalability of relational database management systems.

References

Books:

[Hansen 1996, pg 432]
Hansen, G.W., and Hansen, J.V., *Database Management and Design*, 1996, Prentice Hall

[Rob *et al*, 2002, pg 448]
Rob, P. and Coronel, C., *Database Systems: Design, Implementation, & Management*, *4th Ed,* 2002, Thomson Learning - Course Technology, USA.

[Jones *et al*, 1997]
Jones, J. and Monk, S., *Databases in Theory and Practice*, International Thomson Computer Press, 1997

[Chorafas, 1998]
Chorafas, D.N., *Transaction Management: Managing Complex transactions and Sharing Distributed Database*, St.Martin's Press, Inc., 1998

[Post, 1999]
Post, G.V., *Database Management Systems: Designing and Building Business Application*, Irwin/McGraw-Hill, 1999

Online references:

[Burleson, 2002]
Burleson D, August 8 2002."Oracle Guru" *Database Benchmark Wars: What you need to know* [Online]. Available from: http://www.dba-oracle.com/art_db_benchmark.htm . Last Updated 05/08/2002, Access date 15/05/05

[Kevin Kline, 2005]
Kevin Kline, Director of Technology, *White Papers*, 2005. [Online] Available from: http://www.quest.com/documents/list.aspx?searchoff=true&technology=8&contenttypeid=1, Last Updated 2005, Access date 12/06/05.

[Balloni, 2000]

Michael Balloni, 2000. *SQL Server Lock Contention Tamed: the Joys of NOLOCK and ROWLOCK*, [Online] Available from: http://www.sql-server-performance.com/lock_contention_tamed_article.asp Access date 06/06/05

[Microsoft SQL Server documentation, 2005]

Microsoft SQL Server documentation, 2005 *Understanding Locking in SQL Server*, [Online] Available from:
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/acdata/ac_8_con_7a_7xde.asp .
Access date 15/05/05

[Meier *et al*, 1994]

J.D. Meier, Srinath Vasireddy, Ashish Babbar, and Alex Mackman, 2004. *Improving .Net Application performance and scalability* [Online] Available from:
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnpag/html/scalenetchapt16.asp
Last Updated May 2004, Access Date 06/05/05.

[Transaction Processing Performance Council, 2005]

Transaction Processing Performance Council, 2005 *Top Ten TPC-C by Performance* [Online] Available from: http://www.tpc.org/tpcc/default.asp .Last Updated 21-Sept-2005 7:52 PM Access date 14/05/05.

[RPM Solutions Pty Ltd, 2004]

RPM Solutions Pty Ltd, 2004. *Performance Tests* [Online] Available from:
http://www.loadtest.com.au/types_of_tests/performance_tests.htm .Last Updated August 04, 2004 , Access date 06/05/05.

[MSDN library, 2005]

MSDN library for SQL Server Developer Centre, 2005, *Concurrency Problems*, [Online] Available from: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/acdata/ac_8_con_7a_8i93.asp, Last Updated 2005, Access date 15/10/05.

[MSDN library, Using Partitioned Views, 2005]

MSDN library for SQL Server Developer Centre, 2005 *Using Partitioned Views* [Online] Available from: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/acdata/ac_8_con_7a_8i93.asp , Last Updated 2005, Access date 15/10/05.

Appendix A Tutorial 1: *Universal Software Tools*

1.1 System Performance Monitor

System Performance Monitor tool is performance monitoring tool. It is composed of two parts: System Monitor and Performance Logs and Alerts. With System Monitor, you can collect and view real-time data about memory, disk, processor, network, and other activity in graph, histogram, or report form. Through Performance Logs and Alerts you can configure logs to record performance data and set system alerts to notify you when a specified counter's value is above or below a defined threshold [Help and Support Centre]. When used with Quest spotlight they form a very powerful performance monitoring system.

To open Performance Monitor, click -> Start, click-> Control Panel, click-> Performance and Maintenance, click ->Administrative Tools, and then double-click Performance.

1.2 Enterprise Manager

This is the central point from which all component of the server are managed. Using the Enterprise Manager you can have access to performance tuning tools for of the DBMS.

You can use Enterprise Manager to:

- Administer, diagnose, and tune multiple databases.
- Manage a wide range of "targets" in addition to Oracle databases, including: Web servers, application servers, applications, and Microsoft® SQL Server.
- Monitor target conditions throughout the network
- Create, schedule, and publish HTML reports to quickly view and analyze information about your managed systems.
- Automate repetitive tasks on multiple targets at varying time intervals

To open Enterprise Manager for:

Oracle 9i, click -> Start, click-> All Programs, click->Oracle- <name of your Host>, click-> Enterprise Management Packs, and then click-> Enterprise Manager

Username: phathi

Password: phathisile

SQL Server 2000, click -> Start, click-> All Programs, click-> Microsoft SQL Server, click-> Enterprise Manager

There are no passwords needed

Appendix B Tutorial 2: *SQL Server 2000 Specific Tools*

2.1 SQL Server 2000 Query Analyzer

The SQL Server 2000 Query Analyzer is not only a tool for developing and debugging Transact-SQL code, it is also a great tool for performance tuning Transact-SQL code. Another of its powerful functionality which I found useful to my research is its ability to display the execution plan followed by the SQL Optimizer to actually execute the queries. Added to the graphical view if you move the cursor on top of each of the steps in the query plan, a pop-up box appears with detailed information exactly what the Query Optimizer did in each step as the query was executed as well as the estimated time of execution.

Show Server Trace can be used to help performance tune queries, stored procedures and Transact-SQL scripts. What it does is display the communications sent from the Query Analyzer (acting as a SQL Server client) to SQL Server. The results of the trace are in the form of rows, with each row representing a distinct communication from Query Analyzer to SQL Server. Each row includes the text of the communication, such as Transact-SQL code; the Event Class, which describes the type of communication being sent; the duration of the communication; the amount of CPU time used, and how many reads or writes that were performed for the event. This information can be very valuable when analyzing query performance, and when comparing the performance of one variation of a query against another.

Show Client Statistics Like the Show Server Trace feature, the Show Client Statistics can be very helpful when performance tuning queries, stored procedures, and scripts. What this option does is provide you with application profile, network, and time statistics of whatever Transact-SQL you are running in Query Analyzer. This statistics provide additional information you can use to see how efficiently a query is running, and also allows you to easily compare one query against another.

Manage Statistics Without you doing anything, SQL Server automatically creates and maintains internal statistics on the rows of data in all of your tables. These statistics are used by the Query Optimizer to select the optimal execution plan of Transact-SQL code. Most of the time, SQL Server does a fine job of maintaining these statistics, and the Query Optimizer has the necessary information it needs to do its job.

To open SQL Query Analyser, click -> Start, click-> All Programs, click-> Microsoft SQL Server, click-> Enterprise Manager, on the Tools menu, click-> SQL Profiler

2.2 SQL Server 2000 Profiler

The SQL Server 2000 Profiler is a powerful tool for helping identify SQL Server performance problems, but it is not a tool for the beginner. Essentially, it allows you to capture the communications between your application and SQL Server. The SQL Server 2000 Profiler can capture virtually all communication between a SQL Server and any other application. The various communications you can capture are referred to as events, and are grouped in Event Classes.

Use SQL Profiler to:

- Monitor the performance of an instance of SQL Server. This was the most important aspect for this project.
- Debug Transact-SQL statements and stored procedures.
- Identify slow-executing queries during Performance Testing
- Test SQL statements and stored procedures in the development phase of a project by single-stepping through statements to confirm that the code works as expected.
- Above all the Profiler provides performance statistical information and produce execution plan for analysis.

To open SQL Server 2000 Profiler , click -> Start, click-> All Programs, click-> Microsoft SQL Server, click-> Enterprise Manager, on the Tools menu, click-> SQL Profiler

Appendix C Tutorial 3: *Oracle 9i Specific Tools*

3.1 Oracle 9i Statspack

Statspack is primarily a diagnostic tool for instance-wide performance problems; it also supports application tuning activities by identifying high-load SQL statements. It can be used both proactively to monitor the changing load on a system, and also reactively to investigate a performance problem (this is why it was chosen statistical performance and scalability testing in the project).

Using Statspack: Installing Statspack

Run the installation script using SQL*Plus from within the $ORACLE_HOME/rdbms/admin directory:

```
SQL> connect / as sysdba
SQL> @spcreate
```

This will create a database user PERFSTAT and the Statspack schema. Note that the script must be run from SQL*Plus. All subsequent Statspack operations are run as the PERFSTAT user.

Data collection

Once you have installed Statspack the simplest way to collect a ‘snapshot’ of performance data is by executing the snap function from the Statspack package.

```
SQL> execute statspack.snap;
```

Data collection can also be automated at either the OS or database level using the dbms_job package.

Producing reports

To examine the delta in statistics between two times, run the spreport.sql report while being connected to the PERFSTAT user.
You will be prompted for:
1. The beginning snapshot Id
2. The ending snapshot Id
3. The name of the report output file to be created (default name includes the begin and end snapshot id)

This will produce reports similar to the ones shown in figures 3-7 and 3-8

To open Statspack package, click -> Start, click-> All Programs, click-> Oracle, click-> Application Development, SQL Plus and then connect as shown above.

Appendix D Tutorial 4: *Third party software*

4.1 Quest Central for Oracle 9i and SQL server 2000 and Quest Spotlight

Quest Central for Oracle or SQL Server is an integrated database management solution that is designed to simplify everyday DBA tasks. Whether you are doing an intensive analysis or simply tuning your database, Quest Central finds and resolves thousands of bottlenecks both historically and in real time for optimum performance and availability. [www.Quest.com]

Quest Central for Oracle components include:

- Database Administration
- Space Management
- 24x7 monitoring
- Performance Diagnostics with Spotlight
- Database analysis (health check)
- SQL Tuning
- Performance Analysis
- Load Testing and Data Generation

To open Quest Central and Quest Spotlight, click -> Start, click-> All Programs, click->Quest Software, click-> Spotlight or Quest Central

Using Quest Central and Spotlight

The version used for this project was a complete trail version with a licence valid for 7 days which could be renew on request. Benchmark factory is a 30 days trail version.

The testing environment presented in chapter 2 [Figure 2-1] shows that you need to install the software on the client machine and on the server side you need to install a small application program provided with the down loads which Oracle uses as an interface between it and the client machine. Once the installations are finished you do not need any complex password, you are generally read to use the software. The software interface in so user friendly that navigation around the tool bars is as easy as just clicking buttons and the rest will follow.

This software is available online from www.quest.com under the section on database solution

Appendix E *Metrics*

Wait Event Categories:

Wait events are statistics that are incremented by a server process/thread to indicate that it had to wait for an event to complete before being able to continue processing. Wait event data reveals various symptoms of problems that might be degrading performance within the system. The following wait event categories were considered in calculation the final load and stress testing results. All wait event measurements are in *seconds.*

Wait Event	Description
CPU Usage	When SQL statements and other types of calls are made to SQL Server, an amount of CPU time is necessary to process the call. Average calls require a small amount of CPU time. However, a SQL statement involving a large amount of data or a runaway query can potentially consume a large amount of CPU time, reducing CPU time available for other processing. CPU utilization is the most important operating system statistic in the tuning process. Excessive CPU usage usually means that there is little idle CPU on the system. This could be caused by an inadequately-sized system, by un-tuned SQL statements, or by inefficient application programs.
CPU Wait	Wait Time until the CPU resource is available. Time spent by the session waiting in the system's run queue for CPU cycles. The amount of time is dependant upon the number of concurrent processes and threads requesting CPU time. The metric value should be inspected in conjunction with the value of the "Run Queue Length" metric.
I/O Wait	Time spent waiting for disk input/output operations to complete. Input/output (I/O) is one of the most expensive operations in a database system. SQL statements that are I/O intensive can monopolize memory and disk use and cause other database operations to compete for these resources. Generally, I/O Wait is caused by poorly-tuned SQL or applications which generate a significant amount of logical I/O translating into excessive physical disk usage. In this case, SQL/Application tuning can reduce the logical I/O- induced load. However, it could also be caused by poorly-configured disks or storage sub-systems.
Network Wait	Time spent waiting for messages to be sent or received over the network interface. Network performance can be evaluated by the number (per second) of packets sent and received. Excessive network wait can be caused by either: • excessive network usage originating in the application

	physical problems, identifiable by network errors and network collisions
Lock Wait	Time spent by the session being blocked, waiting for the blocking lock to be released.
Parallel Coordination	The time spent by the various processes coordinating parallel query threads and exchanging data.
Cursor Synchronization	The time spent by the various processes synchronizing information flow within cursors.
Remote Provider	The time spent by the various processes waiting for a remote OLEDB call to complete or DTS synchronization
Log	The time spent waiting by the various processes waiting for a LOG operation to complete.
Other Wait	The aggregate performance of assorted different wait metrics. Elapsed time spent waiting for miscellaneous operations to complete. None of the operations can be classified into any other wait categories.
Latch Wait	Time spent by the session being blocked by a latch, waiting for it to be released. Latches need not be locked for the duration of a transaction. They are low-overhead, short-term memory synchronization objects. They are used mostly to protect a row when queried for a connection.

Complete Metric Listing:

These metrics were used to track numerous aspects of the database system. Some of these include Network latency, Response Time, Throughput, Resource usage (CPU and Memory utilisation, Disk I/O subsystem). These metrics were used during functional, load and stress testing by System Monitor, SQL Server 2000 Query, Profiler, Quest Central, Benchmark factory, stoplight and Oracle 9i Statspack.

Metric	Description
Average Lock Duration (Seconds)	Average duration of blocked lock requests $\frac{100 * \text{Lock Wait}}{\text{Lock Requests}}$
Average SQL Duration (Seconds)	Average duration of the SQL statements, executed during the current interval.
Blocked Lock Requests	Number of lock requests that could not be satisfied immediately, causing the caller to block and wait before acquiring the lock.

Metric	Description
Blocked Lock Requests Ratio	This ratio measures the percentage of lock requests that required the caller to wait before acquiring the lock. High percentage (over 20%) along with high Lock Wait indicates that the system is not handling locks well and concurrency mechanisms need to be tuned. Excessive blocking can be a major cause of poor application performance, as the user of an application often does not realize that they are waiting on a lock held by another user. From their point of view, it often seems like their application has stopped responding. $\frac{100 * \text{Blocked Lock Requests}}{\text{Lock Requests}}$
Cache Hit Ratio (%cache hits)	The percentage of pages that were found in the buffer pool without having to incur a read from disk. When this percentage is high your server is operating at optimal efficiency (as far as disk I/O is concerned). A low Buffer Cache hit rate indicates that SQL Server is finding fewer pages already in memory, and therefore has to perform more disk reads. This is often caused by one of two possibilities: SQL Server has insufficient memory to work with, or SQL queries are accessing a very large number of pages in a non-sequential manner. The best figure will vary from one application to another, but ideally this ratio should be above 90%.
DB CPU Usage (Seconds)	Total amount of CPU consumption as reported by SQL Server in the system processes table.
Degree of Parallelism	Average number of SQL Server threads assigned to serve a SQL statement.
Disk Queue Length	Number of I/O requests that were outstanding on the busiest disk in the system.
Disk Utilization	The percentage of time the busiest disk spent serving system-wide I/O requests.
File System Cache Hit Ratio (% Cache Hits)	The percentage of file read operations that were satisfied by the file system cache and did not require any physical I/O.
Full Scans	Number of unrestricted full scans. Refers to both table and full index scans.
Full Text Search CPU Usage (% CPU Busy)	CPU consumption of the Full Text Search service.
Full Text Search Resident Memory Usage (MB)	Amount of physical memory consumed by the Full Text Search service.

Index Searches	Number of index searches. Index searches are used to start range scans, single index record fetches, and to reposition within an index.
Lazy Writes	Number of buffers written by the Buffer Managers lazy writer. The Lazy Writer Process periodically scans all SQL Server caches, and maintains a list of "free" pages that are available for immediate reuse. When SQL Server needs a free memory page (for example, when reading a database page from disk into the Buffer Cache), and there are no free pages immediately available, the connection needing the free page must wait while SQL Server makes buffers available. This will result in slower performance. In the worst case, the connection will have to wait while SQL Server writes a modified page out to disk in order to make a free buffer.
Lock Requests	Number of new locks and lock conversions requested from the lock manager.
Log Flushes	Number of log flushes. As users make modifications to SQL Server databases, SQL Server records these changes in a memory structure called the Log Cache. Each SQL Server database has its own log cache. When a user transaction is committed (either explicitly via a COMMIT statement, or implicitly), SQL Server writes all changes from the Log Cache out to the log files on disk. This process is termed a log flush. The user that issued the commit must wait until the log flush is complete before they can continue. If the log flushes takes a long time, this will degrade the user's response time.
Memory Utilization (% Memory Used)	Total amount of physical RAM consumed by the various processes. High utilization (over 90%) along with high swapping and paging rates indicates that the amount of physical RAM should be increased.
Network Collisions	Number of times when two computers send packets at the same time on the network and the packets "collide" so that both packets must be re-transmitted.
Network Incoming Traffic (KB)	Volume of data received on all network interfaces.
Network Outgoing Traffic (KB)	Volume of data sent on all network interfaces.
Network Traffic Volume (KB)	The total of incoming and outgoing network traffic.
Non-SQL Server CPU Usage (% CPU Busy)	Overall CPU consumption not associated with the monitored SQL Server instance.
Non-SQL Server	Overall resident memory consumption not associated with the monitored

Resident Memory Usage (% CPU Busy)	SQL Server instance (background and foreground processes).
OLAP CPU Usage (% CPU Busy)	CPU consumption of the Analysis Services processes.
OLAP Resident Memory Usage	Amount of physical memory consumed by the Analysis Services processes.
Parallel Coordination	The time spent by the various processes coordinating parallel query threads and exchanging data.
Physical I/O	Number of physical I/O operations performed.
Run Queue Length (Processes/Threads)	System average run queue. The CPU run queue is a holding area for threads and processes that require the CPU when the CPU is busy serving other processes. The run queue length is an indicator of whether the system has sufficient CPU resources for all the processes it executes. High values along with high CPU utilization, indicates that the system requires faster or more CPUs to handle the given load.
SQL Agent CPU Usage (% CPU Busy)	CPU consumption of the SQL Agent service (related to the monitored instance).
SQL Compilations	Number of SQL compilations.
SQL Executions	Number of statements executed during the current interval.
SQL Executions Ended	Number of statements whose activity finished during the current interval.
SQL Executions Started	Number of statements started running during the current interval.
SQL Recompilations	Number of SQL re-compilations.
SQL Server Background CPU Usage (MB)	Amount of memory consumed by the monitored SQL Server instance background processes, both physical and swap memory.
SQL Server Cache Memory (MB)	Total amount of dynamic memory the server is using for the dynamic SQL cache.
SQL Server Connection Memory (MB)	Total amount of dynamic memory the server is using for maintaining connections.
SQL Server CPU Usage (MB)	Overall SQL Server processes CPU usage. SQL Server Background CPU Usage + SQL Server Foreground CPU

	Usage
SQL Server Foreground CPU Usage (% CPU Busy)	CPU time consumed by the monitored SQL Server instance user (session) processes.
SQL Server Free Memory	Total number of free pages on all free lists.
SQL Server Memory Growth Pressure (%)	Relative amount of additional memory (measure in percents) that SQL Server is willing to consume. Values higher than 20% might indicate that the system is low on physical memory. $100 - \frac{100 * \text{Total SQL Instance Memory}}{\text{Target SQL Instance Memory}}$
SQL Server Optimizer Memory (MB)	Total amount of dynamic memory the server is using for query optimization.
SQL Server Physical I/O Operations	Total number of pages read or written to disk. These operations require physical disk access (although some may be satisfied by the file system cache). Page Reads + Page Writes
SQL Server Procedure Cache Memory	Number of pages used to store compiled queries.
Table Lock Escalations	Number of times a lock on a table has been escalated.
SQL Server Swap Memory Usage (MB)	The amount of virtual memory that the SQL Server process has reserved for use in the paging file(s).
Target SQL Server Memory (MB)	Total amount of dynamic memory the server is willing to consume. A value significantly higher than Total SQL Instance Memory might indicate that the system is low on physical memory.
SQL Server/System CPU Usage (% CPU Busy)	The amount of CPU consumed by the entire system broken into SQL Server and Non-MSSQL activity. High values indicate that SQL Server is a major CPU consumer and is likely causing bottlenecks. It is thus the object of tuning efforts. Low values indicate other system activity as the major CPU consumer and the problem solution likely resides with other applications or at the operating system level.
Total CPU Usage (MB)	Overall operating system CPU Usage (including SQL Server). Total Kernel CPU Usage + Total User CPU Usage
Total Free Memory	Amount of free RAM in the system.

(MB)	
Total Kernel CPU Usage	Percentage of CPU time consumed by the operating system's processes (kernel mode activities).
Total Logical Reads	Number of system-wide logical reads. The metric counts both Physical I/O reads which require disk access and reads which were satisfied entirely by the operating systems file cache.
Total Logical Writes	Number of system-wide logical writes. The metric counts both Physical I/O writes that require disk access and writes performed entirely in the operating system's file cache.
Total Memory Usage (MB)	Amount of memory consumed by entire OS processes (including SQL Server), both RAM resident and swapped.
Total Physical I/O Operations	Total number of disk operations (both read and write) performed by the operating system. This metric does not include operations that were satisfied using the file system cache. Total Physical Reads + Total Physical Writes
Total Physical Writes	Number of system-wide physical writes. The metric counts Physical I/O writes that require disk access.
Total User CPU Usage	Percentage of CPU time consumed by the operating system's processes (user mode activities).
Total SQL Server Memory (MB)	Amount of memory consumed by SQL Server, both RAM resident and swapped.

Appendix F: General Information and Scripts

The information and experiments provided in this project are applicable to any database in both Oracles 9i and SQL Server 2000.

All scripts and the database used for this project are provided on the CD with all other relevant information.

There are no passwords needed to re-execute any part of this project except that when you are logging on to Oracle Enterprise Manager one has to be the Administrator otherwise you will not have all necessary rights.

Below are same of the important scripts used for the creation of database schema, indexes and an UPDATE/DELETE script used for same scenarios**.**

Declaimer: These scripts are very dangerous thus use at own risk

-- Oracle 9i PerformanceTestingDatabase create constraints and indices script

```
-- Oracle9i version
-- note: don't use "compute statistics" option; that will be done later

echo on
connect PerformanceTestingDatabase/g05s5782

select 'starting full indexing run' as message, to_char(sysdate,'MM-DD-YYYY
HH24:MI:SS') as time from dual;

select 'creating customer indices' as message, to_char(sysdate,'MM-DD-YYYY
HH24:MI:SS') as time from dual;

create index customer_username on customer (username) tablespace indx parallel;

select 'finished creating customer indices' as message, to_char(sysdate,'MM-DD-
YYYY HH24:MI:SS') as time from dual;

select 'creating subjects indices' as message, to_char(sysdate,'MM-DD-YYYY
HH24:MI:SS') as time from dual;

create index subjects_subjectname on subjects (subjectname) tablespace indx
parallel;

select 'finished creating subjects indices' as message, to_char(sysdate,'MM-DD-
YYYY HH24:MI:SS') as time from dual;

select 'creating products indices' as message, to_char(sysdate,'MM-DD-YYYY
HH24:MI:SS') as time from dual;

 create index products_subjectid on products (subjectid) tablespace indx
parallel;

create index products_booktitle on products (booktitle) tablespace indx
parallel;
```

```
      create index products_author on products (author) tablespace indx
parallel;

select 'finished creating products indices' as message, to_char(sysdate,'MM-DD-
YYYY HH24:MI:SS') as time from dual;



select 'creating ordercluster indices' as message, to_char(sysdate,'MM-DD-YYYY
HH24:MI:SS') as time from dual;

      create index ordercluster_orderid on cluster ordercluster tablespace indx
parallel;

select 'finished creating ordercluster indices' as message, to_char(sysdate,'MM-
DD-YYYY HH24:MI:SS') as time from dual;

select 'creating orders indices' as message, to_char(sysdate,'MM-DD-YYYY
HH24:MI:SS') as time from dual;



select 'finished creating orders indices' as message, to_char(sysdate,'MM-DD-
YYYY HH24:MI:SS') as time from dual;

select 'creating orderdetails indices' as message, to_char(sysdate,'MM-DD-YYYY
HH24:MI:SS') as time from dual;


select 'finished creating orderdetails indices' as message, to_char(sysdate,'MM-
DD-YYYY HH24:MI:SS') as time from dual;

select 'creating shoppingcart indices' as message, to_char(sysdate,'MM-DD-YYYY
HH24:MI:SS') as time from dual;



      alter table shoppingcart add constraint shoppingcart_customerid_FK foreign
key (customerid) references customer (customerid) parallel;
      alter table shoppingcart add constraint shoppingcart_bookid_FK foreign key
(bookid) references products (bookid) parallel;

select 'finished creating shoppingcart indices' as message, to_char(sysdate,'MM-
DD-YYYY HH24:MI:SS') as time from dual;


select 'finished full indexing run' as message, to_char(sysdate,'MM-DD-YYYY
HH24:MI:SS') as time from dual;

commit;
exit

--------------------------------------------------------------------------------
----
```

-- Oracle *9i PerformanceTestingDatabase create Schema creation Script*

```
set echo on

-- log in as correct user
connect PerformanceTestingDatabase/g05s5782
```

```
-- drop objects
drop table shoppingcart;
drop table ordersdetails;
drop table orders;
drop table products;
drop table subjects;
drop table customer;
drop sequence customer_customerid_seq;
drop sequence orders_orderid_seq;
drop sequence orderdetails_orderitemid_seq;

-- create sequences
-- 10K scale factor: use "create sequence customer_customerid_seq start with
10001;"
-- 5M scale factor: use "create sequence customer_customerid_seq start with
5000001;"
create sequence customer_customerid_seq start with 5000001;
create sequence orders_orderid_seq;
create sequence orderdetails_orderitemid_seq;

-- create tables

-- behavior: read-only table
create table customer (
      customerid int NOT NULL,
      firstname varchar(30) NOT NULL,
      lastname varchar(30) NOT NULL,
      address1 varchar(30),
      address2 varchar(30),
      city varchar(30),
      state varchar(2),
      zip varchar(5),
      email varchar(50),
      phone varchar(20),
      creditcard varchar(30),
      creditcardexpiration varchar(4),
      username varchar(30) NOT NULL,
      password varchar(30) NOT NULL,
      constraint customer_customerid_PK primary key (customerid)
) organization index
  pctfree 0
  tablespace users;

-- behavior: read-only table
create table subjects (
      subjectid int NOT NULL,
      subjectname varchar(30) NOT NULL,
      constraint subjects_subjectid_PK primary key (subjectid)
) organization index
  pctfree 0
  tablespace users;

-- behavior: read-only table
create table products (
      bookid int NOT NULL,
      subjectid int NOT NULL,
      booktitle varchar(50) NOT NULL,
      author varchar(50) NOT NULL,
      price numeric(8,4) NOT NULL,
      retail numeric(8,4) NOT NULL,
      isbn varchar(20) NOT NULL,
      quantityonhand int NOT NULL,
      constraint products_subjectid_bookid_PK primary key (subjectid,bookid)
) organization index
```

```
  compress 1
  pctfree 0
  tablespace users;

-- heap organized (all inserts will go at end of heap)
-- behavior: high insert rate, all of which is at the end of the table; no
updates or deletes
create table orders (
      orderid int NOT NULL,
      customerid int NOT NULL,
      orderdate date default CURRENT_DATE NOT NULL,
      netamount numeric(8,4) NOT NULL,
      tax numeric(8,4) NOT NULL,
      totalamount numeric(8,4) NOT NULL
) pctused 40 pctfree 5 initrans 2
  tablespace users
  storage (initial 5M next 5M pctincrease 0 freelists 10);

-- heap organized (all inserts will go at end of heap)
-- behavior: high insert rate, all of which is at the end of the table; no
updates or deletes
create table orderdetails (
      orderid int NOT NULL,
      orderitemid int NOT NULL,
      bookid int NOT NULL,
      quantity int NOT NULL
) pctused 40 pctfree 5 initrans 2
  tablespace users
  storage (initial 10M next 10M pctincrease 0 freelists 10);

-- behavior: high select, insert and delete rate throughout the table, but the
total number of rows does not grow above a few thousand; updates do not change
the size of the row
create table shoppingcart (
      customerid int NOT NULL,
      bookid int NOT NULL,
      quantity int NOT NULL,
      constraint shoppingcart_custid_bookid_PK primary key (customerid, bookid)
) organization index
  pctfree 5 initrans 2
  tablespace users
  storage (initial 5M next 5M pctincrease 0 freelists 10);

-------------------------
-- end
-------------------------

commit;
exit

--------------------------------------------------------------------------------------------------------------------
----
```

-- Microsoft SQL Server 2000 version
-- create constraints and indices script

```
-- start
use PerformanceTestDatabase
go

----------
-- create data table indices

select "starting full indexing run", getdate()
```

```sql
go

-- customer index
create unique clustered index customer_customerid_clu on customer (customerid)
with fillfactor = 100
go
create unique index customer_username on customer (username) with fillfactor =
100
go

--subject index

create unique clustered index subjects_subjectid_clu on subjects (subjectid)
with fillfactor = 100
go
create index subjects_subjectname on subjects (subjectname) with fillfactor =
100
go

-- products index

create unique clustered index products_subjectid_bookid_clu on products
(subjectid, bookid) with fillfactor = 100
go
create unique index products_bookid on products (bookid)
create index products_booktitle on products (booktitle)
create index products_author on products (author)
go

-- fully covered index for specials listing on homepage
create index products_specials on products (subjectid, bookid, booktitle,
author) with fillfactor = 100
go
-- fully covered index for browse query on subjectid or subject name
create index products_searchsubject on products (subjectid, booktitle, author,
price, retail, bookid) with fillfactor = 100
go
-- fully covered index for browse query on author name
create index products_searchauthor on products (author, retail, price,
booktitle, bookid) with fillfactor = 100
go

-- orders

-- don't use a clustered index -- it leads to contention on the last index page
as new rows are inserted

create unique index orders_orderid on orders (orderid)
go

-- orderdetail

create unique clustered index orderdetail_clu on orderdetails
(orderid,orderitemid)
go

-- shoppingcart

create unique clustered index shoppingcart_clu on shoppingcart (customerid,
bookid)
go

----------
-- add constraints
-- if there wasn't already an index on products(subjectid), this would create
one
alter table products     add        foreign key (subjectid) references subjects
(subjectid)
go

alter table orders       add        foreign key (customerid) references customer
(customerid)
go

alter table orderdetails add        foreign key (orderid) references orders
(orderid)
go
alter table orderdetails add        foreign key (bookid) references products
(bookid)
go

alter table shoppingcart add        foreign key (customerid) references customer
(customerid)
go
alter table shoppingcart add        foreign key (bookid) references products
(bookid)
go

-- end
select "finished full indexing run", getdate()
go
checkpoint
go

--------------------------------------------------------------------------------
----
```

-- Microsoft SQL Server 2000 version
-- create tables

```
-- start
use PerformanceTestDatabase
go

----------
-- create data tables

-- customer

create table customer (
      customerid int not null,
      firstname varchar(30) not null,
      lastname varchar(30) not null,
      address1 varchar(30),
      address2 varchar(30),
      city varchar(30),
      state varchar(2),
      zip varchar(5),
      email varchar(50),
      phone varchar(20),
      creditcard varchar(30),
      creditcardexpiration varchar(4),
      username varchar(30) not null,
      password varchar(30) not null
)
go
```

```
-- subjects

create table subjects (
      subjectid int not null,
      subjectname varchar(30) not null
)
go

-- products

create table products (
      bookid int not null,
      subjectid int not null,
      booktitle varchar(50) not null,
      author varchar(50) not null,
      price money not null,
      retail money not null,
      isbn varchar(20) not null,
      quantityonhand int not null
)
go

-- orders

create table orders (
      orderid numeric(9,0) identity not null,
      customerid int not null,
      orderdate datetime not null,
      netamount money not null,
      tax money not null,
      totalamount money not null

)
go

-- create default_orderdate function to set default value for orders.orderdate

create default default_orderdate as getdate()
go
sp_bindefault default_orderdate,"orders.orderdate"
go

-- orderdetails

create table orderdetails (
      orderid int not null,
      orderitemid numeric(9,0) identity not null,
      bookid int not null,
      quantity int not null,
)
go

-- shoppingcart

create table shoppingcart (
      customerid int not null,
      bookid int not null,
      quantity int not null,
)
go

-- end
checkpoint
```

```
go
```

-- Microsoft SQL Server 2000 version/ Oracle 9i

Purpose: Example: **UPDATE/DELETE** in a loop and commit very X records
           Handy for huge tables that cause rollback segment problems
            DON'T ISSUE COMMIT TOO FREQUENTLY!

```
declare
  i number := 0;
  cursor s1 is SELECT rowid, t.* FROM tab1 t WHERE col1 = 'value1';
begin
  for c1 in s1 loop
      update tab1 set col1 = 'value2'
             where rowid = c1.rowid;

      i := i + 1;              -- Commit after every X records
      if i > 10000 then
         commit;
         i := 0;
      end if;

  end loop;
  commit;
end;
/

-- Note: More advanced users can use the mod() function to commit every N rows.
--       No counter variable required:
--
-- if mod(i, 10000)
--    commit;
--    dbms_output.put_line('Commit issued for rows up to: '||c1%rowcount);
--  end if;
--
```