Design of a Performance Evaluation Tool for Multimedia Databases with Special Reference to Oracle

Submitted in fulfilment of the requirements for the degree of

MASTER OF SCIENCE

By

Tonia Stakemire

Computer Science Department
Rhodes University
Grahamstown
South Africa

April 2003

ABSTRACT

Increased production and use of multimedia data has led to the development of a more advanced Database Management System (DBMS), like an Object Relational Database Management System (ORDBMS). These advanced databases are necessitated by the complexity in structure and the functionality required by multimedia data. Unfortunately, no suitable benchmarks exist with which to test the performance of databases when handling multimedia data. This thesis describes the design of a benchmark to measure the performance of basic functionality found in multimedia databases.

The benchmark, called MORD (Multimedia Object Relational Databases), targets Oracle, a well known commercial Object Relational Database Management System (ORDBMS) that can handle multimedia data. Although MORD targets Oracle, it can easily be applied to other Multimedia Database Management System (MMDBMS) as a result of a design that stressed its portability, and simplicity. MORD consists of a database schema, test data, and code to simulate representative queries on multimedia databases.

A number of experiments are described that validate MORD and ensure its correct design and that its objectives are met. A by-product of these experiments is an initial understanding of the performance of multimedia databases. The experiments show that with multimedia data the buffer cache should be at least large enough to hold the largest dataset, a bigger block size improves the performance, and turning off logging and caching for bulk loading improves the performance. MORD can be used to compare different ORDBMS or to assist in the configuration of a specific database.

ACKNOWLEDGEMENTS

I would like to thank my supervisors, Professor David Sewry and Alfredo Terzoli, for their continuous effort and support throughout the duration of my degree. I would also like to thank Stefan Krueger for his encouragement and for proof reading various drafts of the thesis. Finally, I would like to thank the members of the Rhodes University Computer Science Department and my friends who have helped in many different ways and have made my time here enjoyable.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1 : Introduction

Chapter 1 introduces the work done in this thesis. This chapter describes previous work on multimedia databases, revealing that the performance of such databases can be problematic. It also explains that the Computer Science Department at Rhodes University requires a multimedia application and wants to avoid performance problems. A suitable benchmark is thus required to achieve this. Unfortunately, such a multimedia benchmark was not found and as a result the aim of this thesis is to design one.

1.1 Motivation

Multimedia data, such as audio and video data, is often difficult to manage, but with the rapid increase in the production and use of such data, it is becoming necessary to design and implement applications to support it. A prime example where it was necessary to design an application to support a large amount of multimedia data was the 2001 Highway Africa Conference, which is an annual conference held in South Africa by the Journalism Department of Rhodes University. At this conference a large quantity of multimedia data was produced in the form of photographs, articles, audio and video recordings for the use by delegates and in the daily newspaper and website [HA01]. A Multimedia Database Management System (MMDBMS) supported this data, which is an enhancement of the standard Relational Database Management System (RDBMS), as it offers many advantages over alternative options. Examples of these advantages include those of databases, such as data consistency, integrity control, and the elimination of redundancy, in addition to more advanced data modelling capabilities.

The multimedia database was designed, setup, and tested in the several months prior to the conference. The task of testing it was difficult, due mainly to the lack of suitable testing tools, as well as insufficient time and knowledge for their design. When the application was put to use during the week long conference, many unexpected problems were found. Of these problems, the most significant one was the fact that the system did not respond fast enough to the users' requests. To ensure that such problems do not recur, proper testing is essential for future implementations of multimedia databases.

The Computer Science Department at Rhodes University has a similar situation to the one of the Highway Africa Conference in that it produces a large amount of multimedia data that needs to be managed. This multimedia data is predominantly produced by the Centre of Excellence in Distributed Multimedia, and other research groups, such as the ones in virtual reality and audio engineering. These groups produce a large quantity of multimedia data, each with different formats and sizes, as well as unique user requirements. It was again decided that a multimedia database should be used to store the data in an organised manner. To avoid the problems encountered with the multimedia database designed for the Highway Africa Conference, a tool was required to assist in testing the performance of the database before putting it into operation. This tool can be applied to a MMDBMS to help identify and eliminate the most significant performance problems. Only fine tuning will then be needed when the final system is implemented.

1.2 Multimedia Databases and Their Performance

A MMDBMS can be defined as a DBMS that is able to store multimedia data internally and provide the necessary functionality to manipulate it. The complexity of multimedia data and the advanced functionality required for its handling means that standard DBMS, such as Relational DBMS (RDBMS), the type of database most commonly used, cannot handle multimedia data properly. At best, RDBMS can provide a generic media datatype in the form of Binary Large Objects (BLOBs), and at worst provide as little as a reference to a location where multimedia data is held in the file system provided by the operating system. Since the data in such databases can be altered or deleted without the database even knowing about it, there is a lack of control over the multimedia data. This inability of RDBMS to handle multimedia data adequately has led to the development of a more suitable class of DBMS, the Object Oriented DBMS (OODBMS), which can be extended through the use of object-oriented mechanisms to handle complex data. OODBMS, however, do not have the relational structure found in RDBMS, which manages the data in a simple, yet efficient manner. A hybrid DBMS, which brings together the appealing features of both RDBMSs and OODBMSs, is an Object Relational Database Management Systems (ORDBMS). ORDBMS incorporate the relational structure of RDBMS with the ability of OODBMS to handle complex datatypes. Since ORDBMS are relatively new there is still limited knowledge on how they work and, as with any DBMS, the performance is of particular interest.

System performance can be defined as how well a system executes a task in a given environment. To measure the performance of a system, a mechanism such as a benchmark is often used. A benchmark to measure the performance of a multimedia database should focus on what is common in a multimedia workload. Database benchmarks, including those designed for multimedia, are commonly used to compare different databases, but they can also be used for other purposes. For example, they can be used to assist in the optimal configuration of a database, where configuration is defined as the process of setting up both the physical and logical components of the database. Configuration is closely linked to a task known as tuning, where the configuration of the database is repeatedly modified in order to reach the best performance under certain constraints [KL99].

1.3 Problem Statement

To measure the performance of any aspect of a multimedia database, including the basic functionality such as insert, update and delete, a suitable benchmark is required. Despite the considerable research that has been undertaken to investigate the performance of databases (shown by the large amount of available literature on this topic), none of the resulting benchmarks are suitable for multimedia databases. This is mainly because the workloads (way in which operations are implemented, the type of datasets returned, and the number of users querying the data) of these benchmarks differ from the workloads found in a multimedia application. For example, multimedia workloads, unlike other workloads, require complex queries to perform the basic operations that are performed easily in standard databases. These unique requirements also make it difficult to adapt the existing benchmarks to suit a multimedia database. The problem centres on the design of a new benchmark to test the basic functionality of a multimedia database, which can be developed to measure more advanced multimedia database functionality in the future.

1.4 Aim

The aim of this thesis is to design a suitable benchmark to evaluate the performance of basic functionality, especially insert, update and delete operations, in a multimedia databases. This benchmark is called MORD, which stands for a Multimedia Object Relational Database benchmark, and it targets the Oracle ORDBMS. Although MORD is designed specifically to work on Oracle, it is also designed to be adaptable. It is adaptable since most settings of the benchmark can be changed and the test environment can be altered. For example, the test data used and queries tested by MORD can be altered.

As with any benchmark, MORD need to be validated to ensure that it includes all of the required functionality and that the functionality works correctly. Validation of MORD is performed using a variety of tests run on a selection of Oracle configurations. While validating MORD, it would also be beneficial to obtain a basic understanding of Oracle's performance when dealing with multimedia data. As a result, a secondary goal of testing MORD is to test whether established database theory that holds for standard data also holds for multimedia data. Several informal hypotheses are formed, based on such general database theory, and tested. The testing of these informal hypotheses demonstrates how MORD can be applied, investigates whether it functions as expected, and gives a better understanding of the performance of multimedia applications on Oracle.

1.5 Focus and Scope

The focus of this thesis is on the design of a benchmark to evaluate the performance of basic functionality in multimedia databases. Planning the design of a thesis requires that the decision is made of whether a broad area is studied in less detail or a specific area is concentrated on in finer detail. For this thesis the latter was chosen, meaning that the scope was reduced and MORD could only test a limited area. The options were that MORD either investigated basic multimedia functionality or that it focused on more advanced specialised multimedia functionality, but not both. Basic functionality was selected since there are currently no multimedia database benchmarks (as justified in the next Chapter) and it is not logical to design a benchmark to test advanced multimedia functionality until one has been designed for basic functionality.

The aim was achieved by first conducting a comprehensive study of existing benchmarks and multimedia databases, studying their unique characteristics in particular. MORD was designed based on the findings of this study. Although MORD was subsequently tested and preliminary results were obtained from this, a thorough evaluation of Oracle's performance was not an objective in this thesis. In future, MORD can be applied to multimedia databases more thoroughly to obtain results to assist in the configuration of these databases. For this thesis, the focus is on the design of a benchmark, rather than carrying out an extensive performance evaluation of a specific database.

MORD is designed to be useful in several different multimedia database environments, where its scope is determined by a few factors. These factors include: the number of users, the type and amount of test data, and the kind of operations tested.

MORD's aim is to measure the fundamental performance of multimedia databases. This requires a simplified environment and as a result all of the experiments are executed in a single-user mode.

The amount of test data included with MORD is fairly small, though it is easy to increase it if necessary. The data consists of a selection of images, audio, and video data, and a limited selection of documents. Although documents are generally classified as multimedia data, the main reason for not including a larger collection of them is that they are used in special tests designed to demonstrate that MORD's test data can easily be extended to include a different datatype.

The operations tested by MORD are primitive queries, such as `insert`, `select`, and `update`, that have been adapted for multimedia data. The time taken to execute some advanced queries, specific to multimedia data, is also measured. These queries were selected to explore the backend processing of multimedia data, and include the extraction of metadata, and the creation of thumbnails from images. The front-end processing, such as the streaming of data or content-based retrieval is not investigated. The investigation of advanced queries that are found in standard RDBMS, such as `join` and `aggregate`, is also not performed as they are less relevant in a multimedia environment. Such queries are usually used to either combine data or perform calculations on it, such as calculating the average value, neither of which is generally relevant to multimedia data.

The performance metrics that are used to measure the times for the experiments include response time and throughput, while alternative metrics, such as resource utilisation, are not considered. They are not relevant to the department and are becoming less important as a result of the rapid advances in hardware.

1.6 Organisation of Thesis

The next chapter presents the findings of a literature survey on multimedia databases and database performance. It summarises relevant work on multimedia database to illustrate their significant issues and problem areas. Following this, the work that has been done on the configuration and tuning of databases is discussed. Finally, it describes work that has been carried out on performance and performance evaluations.

Chapter 3 presents a literature survey on existing benchmarks. This chapter goes into detail about the objectives, design decisions, and characteristics of these benchmarks that make them unsuitable to evaluate the performance of multimedia databases. This chapter also describes the requirements of a multimedia database benchmark.

Chapter 4 introduces Oracle, as it is the multimedia database used in this thesis. This chapter explains how Oracle functions by describing its architecture, its components, and the manner in which it stores the data. Details on the handling of multimedia data by Oracle as well the advanced functionality Oracle provides for this data are presented. The chapter ends by summarising past research on Oracle, and more specifically the sections focussing on multimedia.

Chapter 5 describes the design of MORD, firstly by specifying MORD’s objectives. This is followed by specific details on MORD’s design. These details include a description of the schemas used to create the database tables, MORD’s test data, the functionality that MORD tests, how MORD measures the time for the tests, and the programming language MORD uses.

Chapter 6 details the validation tests performed to demonstrate that MORD functions correctly. This chapter lists MORD’s objectives that need to be tested. This is followed by a brief description of the test environment, including the specification of the hardware as well as the configuration of the database. Details are then given on how the validation experiments have been designed as well as the hypotheses tested that relate to the performance of Oracle.

Chapter 7 presents the results of each of the experiments performed, followed by an informal analysis. This chapter aims to indicate that MORD performs as expected as well as to give an initial understanding of how Oracle performs with respect to multimedia data. The most significant findings as well as unexpected findings are presented again at the end of this chapter to summarise them.

Chapter 8 concludes the work presented in the previous chapters, showing that the objectives of the thesis have been met. It also emphasises the most important achievements in this thesis and includes suggestions for future work.

Chapter 2 : Tuning and Performance Evaluation of Multimedia Databases: A Survey

This chapter presents a survey of work related to this thesis. It includes work on the tuning of multimedia databases, as well as benchmarking. Firstly, it describes the unique requirements of multimedia databases that set them apart from other databases. Details are then given about the main areas to consider when tuning a database. Performance evaluations are discussed next.

2.1 Multimedia Databases

2.1.1 Multimedia Data

The rapid technological advancements in computing have led to an increase in the production and use of multimedia data. Multimedia data differs markedly from standard datatypes in both structure and required functionality. Some of the characteristics listed by Klas and Aberer [KA95] are temporal aspects, media representation, data volume, data modelling, and functionality. The *temporal aspect* refers to the time dependant constraints of multimedia data. Klas and Aberer explain that time is relevant to multimedia data because it closely represents reality, where time is a significant factor. For example, a video is usually a sequence of events that must be shown in the correct order and at the right speed for it to be represented properly. They classify multimedia data as either time dependant, such as audio and video, or time-independent, such as images. *Media representation* refers to the necessity for new datatypes as well as operations. Built-in datatypes, such as alphanumeric data, are not appropriate to represent the complex structure of multimedia data. Klas and Aberer suggest that a modular and efficient representation of different multimedia formats must be supported, in a transparent manner to the application/user. *Data volume* refers to the large amount of data to be processed. *Data modelling* refers to the more advanced indexing techniques, as well as semantic and consistent modelling of abstractions needed for multimedia *functionality,* such as content-based searching.

2.1.2 Multimedia Databases

Multimedia databases have evolved from traditional databases, starting with RDBMS. RDBMS were developed approximately 30 years ago and are currently a popular, well-used technology that can be implemented easily. They facilitate the efficient storage of data through the use of tables controlled by mathematical relations, a concept that originated from work done by Ted Codd in 1970 [CO70]. As a result, they enforce a very rigorous structure with fixed schemas for their databases. With standard datatypes this is advantageous, as they are part of the predetermined set of attributes of RDBMS, but it makes it difficult to extend the datatypes and functionality to support complex datatypes. Examples of RDBMS include: Microsoft SQL Server and Access, IBM's DB2, Sybase, and MySQL.

Object-Oriented DBMS (OODBMS) were developed roughly a decade later, with the intention of overcoming some of RDBMS's problems, such as their inability to be extended to handle more complex datatypes. Inherent in OODBMS is an extendible data model that uses objects, attributes, classes, methods and messages to overcome the inflexibility of RDBMS [SZ96]. Although this allows the specification and management of complex media objects through Abstract Data Types (ADT), unfortunately these datatypes are not already integrated into the system. Another shortcoming of OODBMS is the lack of a standard ad-hoc query language, such as SQL, with which to manipulate the data [STO96]. Examples of OODBMS include: Jasmine, GemStone, Ontos, Objectivity, Versant, and STONE Object Store.

A valuable addition to database technology is the recently developed Object-Relational DBMS (ORDBMS). These DBMS are based on RDBMS technology with the extension of functionality that was previously limited to OODBMS technology. Such functionality includes inheritance, complex object support, an extensible type system, and triggers [STO96, and CAR97]. Stonebraker *et al.* [STO96] suggest that the development of ORDBMS is the most striking advance in DBMS functionality since RDBMS were first introduced. Although ORDBMS are growing in importance, their technology has not reached a stage of maturity yet. A contributing factor to this is the fact that no standard for ORDBMS exists as ANSI

X3H2 have not completed their work on SQL3 [STO96]. Examples of ORDBMS include: Informix, Postgres, and Oracle.

A different approach would be to develop SQL itself rather than the database, and although this would not be considered a 'true multimedia database', it is an interesting alternative for handling multimedia data. SQL/MED, standing for the SQL Management of External Data, is the new addition to the SQL standard, originating in the early part of 2001. J. Melton *et al.* explain that SQL/MED: "offers syntax extensions to SQL as well as a set of routines for use in developing and managing applications that access both SQL and non-SQL (also known as external) data". SQL/MED is divided into two sections; one called the *wrapper interface* and the other called the *datalink* that together manage the external data as if it was stored within the database. The wrapper interface allows the user to view the external data, while the datalink allows the server maintain control of the data, such as integrity control. SQL/MED no longer has the limitation found in RDBMS, as it does not need to have additional datatypes since the data is stored externally. Its functionality can be extended to include the majority of the functionality, such as referential integrity, recovery mechanisms, and authorisation mechanisms [MMJ02], which are not normally found with external data. It still, however, has to be explicitly implemented and is not integrated into the database. There is also the limitation that no distinction can be made between the different multimedia datatypes, and thus specialised functionality, such as cropping images, cannot be implemented.

A specialised database has been designed to support multimedia data, known as a Multimedia DBMS (MMDBMS). It is largely based on OODBMS technology, with extra functionality to support multimedia data. This functionality consists of multimedia datatypes as well as continuous data delivery. MMDBMS may be a viable option in the future, but at present there are no commercial or even widely used MMDBMS. One example is the AMOS system, which is one of the most advanced MMDBMS.

To identify which of the DBMS are suitable for handling multimedia data, it is necessary to compare whether they can support the complex structure and specialised functionality of multimedia data. Such a comparison is given in Table 2-1, where their built-in datatypes to handle multimedia, and their functionality to support basic multimedia queries, such as inserting, as well as advanced multimedia queries, such as streaming, are investigated.

Feature	RDBMS	OODBMS	ORDBMS	SQL/MED	MMDBMS
Multimedia Data Support	Limited to BLOB and References	Extended to support multimedia data	Often built-in otherwise extendible	Extensions in SQL to support multimedia data	Built-in support for all multimedia data
Query Languages	SQL only	OO languages and high level languages only	SQL, high-level and OO languages	Advanced SQL with extensions	Usually OO and high-level languages
Multimedia Functionality	None provided. Cannot be added	Limited multimedia functionality	Often full multimedia functionality	No multimedia functionality	Full multimedia functionality
Usage	Well-used and Mature	Well used, especially for engineering applications	Still fairly new, already extensively used	Not widely used	Used only in research environments

Table 2-1 Comparisons of OODBMS, ORDBMS, SQL/MED, and MMDBMS

RDBMS are unsuitable for multimedia data, given their inability to support complex data and its corresponding functions. Although some RDBMS can support multimedia data by the inclusion of Binary Large Objects (BLOBs) in their set of attributes, they still do not offer the required functionality. OODBMS can be used for multimedia databases, but ORDBMS appear to be the best solution as they integrate the best of both RDBMS and OODBMS technology. The extendibility of ORDBMS allows for multimedia data to be included as a built-in datatype of the system that can then be queried using the simple SQL query language, which is not possible with OODBMS. SQL provides a higher level of protection from programming errors as well as making the data modelling and querying easier. Since ORDBMS appear to be the best overall technology to use for multimedia applications, they are further reviewed briefly.

One of the first types of ORDBMS developed was POSTGRES, originally developed by Berkeley University [STO96]. It is an open source database, initially based on RDBMS technology that has been extended and is continuously being improved. Although it is well suited to handling massive storage and can store large objects, it does not have inherent support for multimedia data.

Following POSTGRES was the development of Illustra, a commercial ORDBMS that was later bought out by Informix and became part of the Informix ORDBMS. Informix allowed users to create manageable and easily distributable multimedia repositories and so became one of the most popular commercial ORDBMS supporting multimedia data. It achieves this through Media360, which runs on Informix's object-relational database, Internet Foundation 2000. Media360 handles multimedia data together with the Informix Dynamic Server/ Universal Data Option (IDS/UD) [INF01]. In 2001, IBM bought Informix and has plans to integrate it with DB2, especially for multimedia applications. Previously, IBM's DB2 only provided user-defined functions, but not user-defined datatypes. This limited its multimedia capabilities since it only has BLOB datatypes and no support for specialised multimedia datatypes. Its latest release includes the DB2 AIV Extender suite, which has image, audio, and video extenders based on technology from Informix's multimedia data management modules previously known as *datablades*. These permit user-defined datatypes and user-defined functionality to support multimedia data that can be queried using SQL statements.

Another popular ORDBMS with multimedia capabilities is Oracle. Oracle 9*i* supports multimedia through an application known as *Inter*Media which incorporates the datatypes in the system. It also provides integrated queries and functionality, such as searching by content, manipulating images and altering the format of the data. At present, Oracle appears to be a viable ORDBMS for handling multimedia data due to its advanced datatypes and multimedia functionality.

2.1.3 Multimedia Applications

There are numerous benefits to the use of multimedia data in an application, including the addition of sound and colour to what may otherwise be a dull application. The saying, "a picture can say more than a thousand words" sums up the power that visual information has, and the addition of sound is no different. This has resulted in the use of multimedia data in applications that vary from police criminal identification systems to educational systems, from systems for the movie industry to home shopping websites, just to name a few [SUB98]. Mittag [MIT00] emphasises the increasing importance of using multimedia databases in education, based on the results produced by his research where he tests its effectiveness. Another area of particular interest is the use of multimedia databases for virtual reality. Soetebier *et al.* [SDB99] proposed a database to store the different components that are needed to construct their virtual reality environment. The benefits of using a database include, according to them, easy and central administration, reusability of components, and seamless integration between the database and their environment.

2.2 Database Tuning and Configuration

2.2.1 Database Tuning

Sasha and Bonnet [SB02] describe database tuning as making the application run faster. This can either mean a greater throughput or a faster response time for time-critical applications.

Database tuning involves identifying where the performance is being slowed down, known as the *bottleneck*, and tuning this.

The tuning of databases should ideally be automatic so that they can adapt to any variation in the workload. Much work is being done in this area, but unfortunately difficulties arise from the fact that each database has a different physical and logical implementation, varied workload, and specific performance requirements [KL99]. As a result, automatic tuning is yet to become a built-in feature of commercial databases and database tuning still has to be performed manually by identifying the bottlenecks and adjusting the configuration of the database in order to eliminate them.

Sasha and Bonnet studied performance tuning and consider the areas illustrated in Figure 2-1as possible bottlenecks.

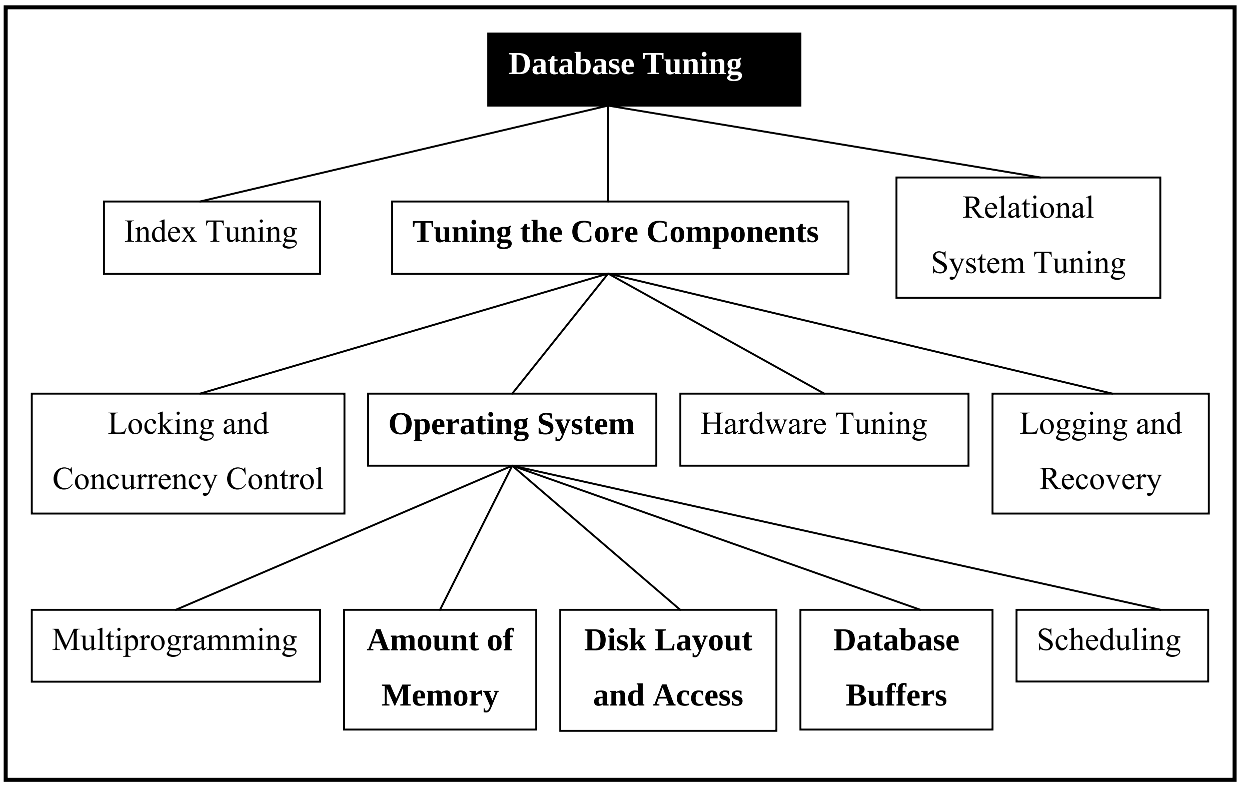


Figure 2-1Outline of database tuning areas

Given the nature of multimedia applications, the most relevant of these tuning areas are the components of the Operating System (OS). Of these, the highlighted areas, including the amount of memory, the disk layout and access, and the database buffers, are the most significant.

2.2.2 Operating System Tuning

Main memory is increasingly becoming the bottleneck for database applications [BMK99]. Boncz *et al.* [BMK99] suggest one reason for this is that the speed of CPUs has advanced rapidly, which means that they are no longer the bottleneck. *Database buffers* are the sections of the main system memory that are reserved for the database's usage. Database buffers that contain the required data reduce the disk Input/Output (I/O), and as a result they significantly improve the performance of a database, as access to main memory is much faster than disk access [ORS96]. The large size of multimedia data (among other reasons) means that reducing the amount of I/O is extremely important, and the amount of memory reserved for the database buffers is crucial for these types of applications.

The subsystem responsible for the allocation of the buffer space, known as the buffer manager, also influences the performance of the database to a large extent [ORS96]. Ozden *et al* [ORS96] studied different replacement algorithms used for multimedia storage systems, and found the currently used approximation algorithms, namely Least Recently Used (LRU) and Most Recently Used (MRU), yielded poor performance.

Storage access refers to the physical mapping and storage of the data on the underlying disk. Storage is implemented by dividing the files in which the data is stored into partitions of fixed length, known as database blocks. To improve the performance of a database, the goal is to reduce the number of blocks transferred between the disk and main memory. Shapiro and Miller [SM99] investigated the tuning of both I/O and memory for databases when using BLOBs. They evaluated the performance of the database with different configurations, finding that through memory and I/O tuning they managed to improve the performance significantly (see section 4.6.2 for more details).

Two other tuning areas of the operating system that are less relevant to multimedia data are multiprogramming and scheduling. Multiprogramming refers to the concurrent execution of queries, where performance is improved in a multi-user environment by simultaneously running queries. This is made possible firstly by enforcing concurrency control, and secondly by assigning the query its required resources as soon as it becomes available. For example, while the CPU is servicing one query, another query might be accessing the disk simultaneously. Scheduling refers to the process of delaying the movement of blocks from the disk to main memory so that it can reorder them in the most logical manner. The more the disk-arm movement of the hard drive is reduced by this technique, the larger the performance improvement. Oracle supports both multiprogramming and scheduling automatically.

2.2.3 Core Component Tuning Areas

Tuning the other core components of the database include issues such as hardware setup, locking and concurrency control, and logging and recovery, as shown in Figure 2-1. Although hardware configuration has a large influence on the performance, it is independent of the database. Locking and concurrency control is relevant only in multi-user environments, while logging and recovery is necessary to restore a database when something goes wrong. Both of these are already supported by Oracle.

2.2.4 Additional Database Tuning Areas

Additional database tuning areas include index tuning and relational system tuning. Index tuning generally improves the performance of a database by reducing the overhead and amount of time needed to search and retrieve data from the database. It is important in a multimedia system, as emphasized by Kornacker [KOR99] who studied the high-performance of extensible indexing in ORDBMS. Indexing is usually automatically implemented in databases in the best possible manner, and thus studying this tuning area is generally less important than other tuning areas. Oracle is one such ORDBMS that automatically provides efficient indexes for multimedia data. Relational system tuning refers to the design of the database schemas, as well as the design of the queries that access the data stored in the database. Relational system tuning is less important with multimedia applications since special code is required to create the schemas and query the data, and thus there is often only a single way for each DBMS to implement them.

2.3 Performance Evaluation

2.3.1 Value of Performance Evaluations

Performance evaluations can be used to assess new algorithms or techniques [CHA95, CDN93, and STO93]. This is achieved by evaluating the new as well as the older or previously used technology, and these results can be used to compare and contrast the two technologies revealing the superior of the two. Performance evaluations can also be used to highlight the weaknesses of a system to ensure its quality [CHA95, and CAR97]. This is achieved by evaluating a single system and using the results to identify where the system performs badly as well as where it excels. Performance evaluations can also be used as a diagnostic tool to assist in the efficient implementation of the system [CHA95, RIS00, and RUN02]. This is achieved by evaluating a single system and altering different parameters to improve the overall performance.

2.3.2 Measuring Performance

Theoretical information, although useful, has its limitations and it is only through empirical tests that a true understanding of database performance can be gained. A benchmark is one mechanism for carrying out performance evaluations empirically. (Benchmarks are discussed in more detail in the next chapter.) Benchmarks use performance metrics of which the main available ones are response time, throughput, price/performance, and resource utilisation. Response time is also known as the query processing power because it is a measure of the speed of individual operations or queries. It is the elapsed time from when the user initiates the query or function until the operation has been completed or committed. For example, the time from when a key is pressed to the time the result is displayed on the screen. Throughput differs from response time in that it is a measure of the overall performance of the system. For example, it can evaluate the amount of data that a database can process on average in one second. Resource utilisation is a measure of how much a particular resource is used, but it is not commonly used for database performance evaluations. Rather, it is of greater importance in the evaluation of a systems' hardware. Response time is important to database performance evaluations, as can be seen by the number of previous evaluations that include it, such as BUCKY, Wisconsin and the Michigan benchmark (These benchmarks are discussed in the next chapter). It is also beneficial to include throughput as it evaluates a database from a different aspect.

2.3.3 Calculating Performance

There are two approaches to presenting the results of a performance evaluation. The first is to generate a single value with which to rank or rate the system, while the second approach is to produce multiple results, which may be results for tens or even hundreds of experiments. A single value is much easier to interpret and sometimes also to calculate, but its use is limited as it does not reveal where the performance is poor. It is usually beneficial only if the performance evaluation is being used to compare different systems with the intention of ranking them. Multiple valued results are more common as they have several uses, including: identifying weaknesses and strengths, comparing systems, and tuning systems. If a benchmark returns a moderate number of values, approximately 10 to 20 results, it has the advantage that it can be used for more than just a comparison and is also not too confusing to interpret.

To calculate the performance of a system, some evaluations use the geometric mean while others use the arithmetic mean and both methods have previously been used in performance evaluations. To select the most appropriate method for a particular situation it is necessary to have a greater understanding of the required evaluation. Jacob and Mudge [JM95] wrote a paper that describes the differences between using geometric, arithmetic, and harmonic means. In simple terms, geometric mean answers the question:

> *"If all the quantities had the same value, what would that value have to be in order to achieve the same product?"*

Arithmetic mean answers the question:

> *"If all the quantities had the same value, what would that value have to be in order to achieve the same total?"*

Harmonic mean is the inverse of the arithmetic mean.

In this paper, they explain which method is most appropriate to use for various calculations by using examples to illustrate what the results would be for each case. They found that using the geometric mean has the advantage that it preserves values across normalisation, but that normalised values must not be averaged. Geometric mean is used, for example, in the evaluation of the performance of hardware [DIX93] since the same results will be obtained when comparing systems, irrespective of which computer's times are used as a normalisation factor [JM95]. The disadvantage of the geometric mean is that it does not preserve total run times, which are generally the values that are of most interest when doing a performance evaluation of a database. For this reason, Jacob and Mudge only consider arithmetic and harmonic means. They recommend that the harmonic mean be used for calculations with rates and the arithmetic mean for calculations with times.

The next decision to make is whether to use a set of times or rates, or a ratio of times or rates. A set of times is a measure of how much *Time* is taken per unit of *Somethings,* while a set of rates is how many *Somethings* are accomplished per *Unit Time*. A ratio differs in that it is a unitless measure which can be calculated using either time or rate. It is used to identify how much faster a system performs. Jacob and Mudge recommend using the *ArithmeticMean(times)* when calculating the response time and using the

HarmonicMean(rates) when calculating throughput. Based on their argument, arithmetic mean is the most suitable method of calculating the performance of databases.

2.4 Summary

The increase in production and use of multimedia data has lead to the need to store and manipulate such data efficiently. The problem is that multimedia data is complex in structure and requires specialised functionality that standard RDBMS cannot support. More advanced DBMS have been developed, of which ORDBMS is currently the most suitable for handling multimedia data. ORDBMS have the simplicity of RDBMS as well as the advanced functionality found in OODBMS.

Databases need to be tuned for them to work optimally, and as such ORDBMS also need to be tuned. Significant database tuning areas include the buffers that form the database memory regions, as well as the storage on and access to the physical disk. To assist in the tuning of the database, a performance evaluation can be carried out using the appropriate tools, such as a benchmark.

Chapter 3 : Benchmarking Database: A Survey

The previous chapter surveyed multimedia databases, and the tuning and evaluation of their performance. This chapter surveys existing database benchmarks. It includes a review of the leading benchmark's objectives, design decisions, and problems. It also describes the requirements of a multimedia database benchmark.

3.1 Overview of Benchmark

A benchmark is a tool for empirically measuring the performance of a system. Database benchmarks are a subset of benchmarks that are designed to measure the performance of specific databases. They consist of database schemas, test data, a workload to represents the users' actions (such as the queries they execute), and a timing mechanism. In this section, existing benchmarks are investigated with particular reference to how each of these components is implemented in the benchmark.

3.2 Existing Benchmarks

Existing database benchmarks can be grouped into four categories: generic, business, engineering, and advanced. An additional category is architecture benchmarks, which are hardware benchmarks and not database benchmarks.

3.2.1 Generic Benchmarks

Generic benchmarks evaluate the performance of features unique to a type of DBMS, for example testing the performance implications of including inheritance in ORDBMS, and not features found in a specific type of application, for example testing the performance of any database with a select intensive workload. These benchmarks are commonly used to determine whether a specific type of DBMS meets the requirements of the user.

3.2.1.1 RDBMS - Wisconsin

The Wisconsin benchmark was designed to test the functionality of RDBMS in a simple yet scientific manner. It has been widely used in numerous performance evaluations [BDT83] as well as acting as a guideline to aid designers in creating new benchmarks [RUN02]. This benchmark has two main objectives. The first is to test all the main components of RDBMS, including selection, projection, joins, modify, and delete, while the second is to be easily understood so that new queries can be added effortlessly. These objectives have made Wisconsin a popular benchmark and an invaluable tool for RDBMS [RUN02]. Its popularity was also due to its use in testing the performance of database systems in an unbiased manner.

Unfortunately this benchmark is not without problems. Foremost amongst these are that it is not representative of "real" applications as it is missing tests for some critical functionality, such as bulk loads and database recovery operations [DEW93]. It was also criticised for testing single user environments only, but this was not found to be totally true because of the lack of support for the multi-user benchmark that was later developed.

3.2.1.2 ORDBMS - BUCKY

BUCKY was designed to evaluate the performance of ORDBMS, as there were no suitable tools at the time [CAR97]. The benchmark focuses on the unique ORDBMS features not found in standard RDBMS. These are row types with inheritance, inter-object references, set-valued attributes, methods of row objects and ADT attributes and their methods. An additional objective of this benchmark is to compare the performance of ORDBMS with RDBMS in order to establish the maturity of ORDBMS technology. ORDBMS schemas and their equivalent RDBMS schemas are provided so that this can be achieved. BUCKY is limited in that it does not have the capability to evaluate real systems. It can only evaluate the performance of the object extensions found in ORDBMS, and not the other functionality required by real applications.

3.2.1.3 OODBMS – 007 and predecessors

Several benchmarks have been designed to test OODBMS, such as the Altair Complex-Object Benchmark (ACOB), Sun benchmark, 007 benchmark, and the HyperModel benchmark [CDN93, CHA95, and RIS00]. ACOB is a simple benchmark, aimed at testing sequential scans, reads and updates of complex objects with the objective of comparing object, page and file server architectures [CHA95]. The other three are not only designed to test OODBMS but also engineering applications in specific, and thus are discussed in section 3.2.4.

3.2.2 Architecture Benchmarks

Architecture benchmarks are designed to evaluate the underlying hardware of a system rather than the database system. These benchmarks are important nevertheless as their design requires similar decisions to those made for database benchmarks.

3.2.2.1 AS^{3}AP

The ANSI SQL Standard Scalable and Portable (AS^{3}AP) benchmark is designed for RDBMS, with a special focus on testing specialised hardware, known as database machines [BOT93]. The AS^{3}AP benchmark consists of both single-user and multi-user experiments with the objective of being more generic than other benchmarks. The single-user tests are based on the Wisconsin benchmark, while the multi-user workload represents a mixture of Online Transaction Processing (OLTP), information retrieval, and long transactions. Additional objectives of this benchmark are scalability and portability, providing a uniform metric, and minimising human effort in its implementation. Although it was designed with these objectives, it has been found to be difficult to transfer to other systems since the database generator and the multi-user tests are in reality not portable.

3.2.2.2 SPEC Benchmark

The Standard Performance Evaluation Cooperation (SPEC) was developed by a consortium of 22 vendor companies who wanted to provide a benchmark that could measure the performance of the entire computer system [DIX93]. This is achieved by enabling the benchmark to run on any system independent of its underlying configuration. For example, the system can have any value for its components such as the memory system, operating system, and clock rate. This benchmark tests single processor and multiprocessor machines, as well as multitasking operations to emulate multi-user workloads. The SPEC benchmark produces a single number, which has the advantage that it can easily be used to rank systems. Unfortunately the usage of the SPEC benchmark is limited as it cannot be used to analyse systems any further. It also has the problem that it calculates its results using geometric mean, which is unsuitable for evaluating the performance of databases, as explained in section 2.3.3.

3.2.2.3 MediaBench

MediaBench tests specialised hardware designed to support multimedia and communication applications [LPM97]. Realistic results were achieved by designing a benchmark that represents multimedia applications as accurately as possible, and does so using a high-level programming language. After designing the benchmark, its results were compared to similar results from SPEC. It was found that MediaBench was valuable since significant statistical differences were shown between the two benchmarks in at least four areas, including the achieved instructions-per-clock, instruction cache hit rate, data cache read hit, and memory bus utilisation.

3.2.2.4 Other Architecture Benchmarks

Other benchmarks designed to test the architecture of computers include Dhrystone, Linpack and Whetstone. Although these benchmarks were popular when first developed [DIX93], they are no longer widely used. Firstly, Dhrystone is designed to test the performance of integers on small machines with simple architecture. This is usually only used for embedded systems, making Dhrystone largely unrepresentative of any realistic system [LPM97]. Although Linpack and Whetstone are better than Dhrystone as they are designed to test floating-point performance [DIX93], they are still too unrepresentative of any realistic system to be widely used.

3.2.3 Business Benchmarks

Business benchmarks evaluate the performance of applications commonly used in business-orientated environments, such as On-Line Analytical Processing (OLAP) and Decision Support Systems (DSS).

3.2.3.1 TPC Suites

The TPC benchmark suites were developed by the Transaction Processing Performance Council [TRA02]. They consist of several industry standard benchmarks for evaluating different types of workloads for various applications on RDBMS. TPC used the approach of developing the benchmarks through a process of designing, revising, formalising, and administering rather than attempting to create an all purpose benchmark at once. This process left some of the earlier benchmarks, namely TPC-A, TPC-B, and TPC-D, obsolete, whereas others which are more advanced, including TPC-C, TPC-H, TPC-R and TPC-W, are still widely used. These benchmarks became obsolete since they are unrepresentative of real applications, and sometimes they are bias towards a single vendor's system making them limited in a fair comparison. TPC-C is an order-entry benchmark representing business applications, while TPC-H and TPC-R both evaluate decision support applications by testing business oriented queries and concurrent data modification [TRA02]. These decision support applications are characterised by a large database and complex queries. For this type of application it is useful to have prior knowledge of the queries, and so TPC-R provides this, making it different from TPC-H [TRA02]. TPC-W differs from the rest of the TPC suite as it consists of business activities that are in an electronic commerce environment.

3.2.4 Engineering Benchmarks

Engineering benchmarks are designed to test engineering applications, which are often designed using OODBMS.

3.2.4.1 Sun Benchmark, 007 and Derivatives

The Sun Benchmark, also known as 001, is based on the EDB benchmark with a few modifications [CHA95]. The HyperMedia Benchmark that is based on the node-and-link graph structure often found in hypertext environments later replaced this benchmark. Following these, a benchmark known as 007, was designed to measure the performance of a wide range of features of OODBMS, such as the speed of pointer traversals, the efficiency of updates and the performance of the query processor [CDN93, and CHA95]. This benchmark only tests a single user environment given that there is little contention for data by other users in engineering applications. The 007 benchmark has the advantage over the earlier benchmarks of testing important functionality which the others missed, such as complex objects and sparse versus dense traversals. Unfortunately it attempts to be too comprehensive and is not concise enough, producing too many numbers, which can be tedious to collect, and confusing to interpret. Another problem with the 007 benchmark is that it mainly covers engineering tools such as Computer Aided Design (CAD) tools and is not applicable to other OODBMS.

3.2.4.2 EDB

Since performance is crucial when dealing with engineering applications such as CAD tools, there is a strong need to benchmark the database systems used to support them [CAT93]. This need brought about the development of the Engineered Database Benchmark (EDB). Although engineering applications differ considerably, the EDB aims to test the operations that Cattell found are most frequently performed. These operations consist of insertion and look up of objects in addition to more complex ad-hoc queries, all in a scalable environment. Additional aims included testing if systems support one or more of the following: caching the database in main memory, avoiding the overhead of query optimisation, using pre-computer links, and making alternative database server architectures available [CHA95]. This benchmark is only implemented for single user environments for the same reason given above. The most significant difference between this benchmark and earlier engineering benchmarks is that it is much simpler to implement and understand as it focuses on basic performance, leaving out functionality that doesn't affect engineering applications, such as manipulating BLOB fields, rather than attempting to be a more comprehensive benchmark [CAT93].

3.2.5 Advanced Database Benchmarks

Advanced database benchmarks evaluate the performance of specialised databases or complex applications. Such benchmarks include those designed for semantic databases, eXtensible Mark-up Language (XML) databases, Geographical Information Systems (GIS), and document retrieval systems.

3.2.5.1 Semantic Benchmark

Rishe *et al.* [RIS00] proposed a benchmark to measure the performance of semantic databases. The focus of this benchmark is on databases that require sparse data, complex inheritance and many-to-many relationships. Their objective was to design a benchmark where a general statement of the problem is given, instead of enforcing a predefined implementation. This has the advantage that the benchmark can be implemented efficiently for any type of database, and as a result it has been run on both semantic and relational databases. Unfortunately, since a predefined implementation is not enforced, this benchmark has the disadvantage that the results are dependant on the method chosen, and thus the results can be distorted.

3.2.5.2 Michigan

Since XML querying has recently become more important, the Michigan benchmark was designed to evaluate the performance of such systems [RUN02]. The objective of the Michigan benchmark is to provide a benchmark for XML databases that is equivalent to the Wisconsin RDBMS benchmark. It focuses on a variety of representative tasks that XML databases can perform. Unlike other XML benchmarks, these tasks are the basic queries such as selections, joins, and aggregations [RUN02], and not those found in a specific application. Michigan also has the fundamental characteristics of relevancy, portability, scalability, and simplicity. This makes Michigan an important tool for designers, especially when attempting to identify bad implementations of XML databases. It does not provide insight to assist in the selection between different products.

3.2.5.3 SEQUOIA 2000

The SEQUOIA 2000 benchmark was designed to provide the Earth Science (ES) community with a method of comparing applications, such as GIS databases [STO93]. The purpose of SEQUOIA 2000 is to provide a baseline case to judge new technologies, as well as to give the database community some insight into the ES needs. When considering ES applications the characteristics that Stonebraker *et al.* identify are a massive size, complex datatypes, and sophisticated searching. These characteristics are not exclusive to ES applications, and so SEQUOIA 2000 has the advantage that it can also be used to benchmark engineering and scientific applications.

3.2.5.4 FTR Benchmark

The emergence of Full-Text document Retrieval systems (FTR) in the late 1980s brought about the need for a benchmark to evaluate these systems [DH91]. The FTR benchmark was developed with the objective of being simple yet still generating a realistic FTR workload. This workload simulates a multi-user environment where requests to search for and retrieve documents from what was considered to be large document databases (1GB or more) are executed. The FTR benchmark has the advantage that it tests content-based searches rather than simple key-word searches.

3.3 Criteria for Benchmark

The criteria that a benchmark must satisfy depend on whether it is generic or domain-specific. Jim Gray's *Benchmark Handbook for Database and Transaction Processing Systems*

[GRA93] appears to be the de facto work on benchmarks and thus is often used as a design guideline. Jim Gray believes that generic benchmarks give a general idea of a system's performance, but that no single metric can measure all applications. There is thus a need for domain-specific benchmarks, for example, a multimedia database benchmark. The rest of this section is based on a domain-specific benchmark. Jim Gray lists the four criteria that should be satisfied by a domain-specific benchmark as: portability, relevancy, scalability, and simplicity. Benchmark designers generally attempt to follow this guideline, as clearly seen in the design of the Michigan benchmark [RUN02].

Portability means that the benchmark is easy to implement and use on many different systems. This criterion is crucial if a benchmark is used to compare systems. For example, SEQUOIA 2000 tested ARCINFO, GRASS, IPW and POSTGRES [STO93], Michigan tested a native XML database and an ORDBMS [RUN02], and BUCKY tested a standard RDBMS and an ORDBMS [CAR97].

Relevancy means that the benchmark must measure typical environments and situations rather than uncommon or insignificant cases. The importance of relevancy is emphasised by the stress that benchmark researchers place on the benchmark being representative of a real application. Benchmarks that closely match real workloads, such as TCP-C, TCP-D, Wisconsin, and SEQUOIA 2000, have been widely accepted. Further evidence of the importance of a relevant benchmark comes from the failure of both the Dhrystone benchmark and the TPC-E benchmark. The former was unrepresentative of any actual workload [LPM97], and the relevancy of the latter lay only in a small scope [RIS00].

Scalability means that the benchmark is applicable to both small and large systems. Although this appears to be one of the less important criteria, as some successful benchmarks do not include this characteristic, it does increase the usefulness of the benchmarks. It makes the benchmark applicable to a wider range of systems, as shown by the TPC suite [TRA02, and STO93], and the Wisconsin Benchmark [DEW93].

Simplicity means that the benchmark is easy to use and extend, and is an important characteristic of a benchmark as shown by the trend of simple benchmarks becoming popular and well used [RUN02]. Examples of such benchmarks are the Sun Benchmark for OODBMS [CDN93], and DebitCredit and Wisconsin for RDBMS [RUN02, BDT83, and CHA95].

Two additional criteria that Horricks *et al.* [HPS00] list as being important are reproducibility and being representative. A benchmark is said to be reproducible if it produces the same results repetitively, and to be representative if its tests cover a significant area of the whole input space.

3.4 Design of Benchmark

The design decisions made for existing benchmarks, primarily domain-specific benchmarks, are reviewed in this section.

3.4.1 Schema Design

The data found in a database may change over time but this is always within the constraints of the schema. The schema controls the overall design of the database and the data must conform to this design, making schemas an important consideration in the design of benchmarks.

The schemas of existing domain-specific benchmarks are modelled after typical applications found in the domain being tested. It is difficult for these schemas to represent all possible models in that domain, and as a result they frequently use a representative subsection of the domain. This is illustrated by the schema of the SEQUOIA 2000 benchmark that is modelled after Earth Science applications [STO93] as shown in Figure 3-1.

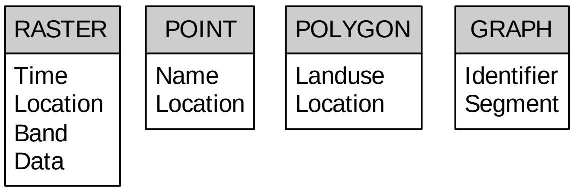


Figure 3-1 SEQUOIA 2000 schema ([STO93])

The table names, such as Raster and Graph, as well as the data in the tables, such as Time and Location, found in the SEQUOIA 2000 schema are unique to Earth Science applications since they describe geographical information. This type of data is not expected in the schemas of other types of applications.

The schemas of engineering benchmarks are generally modelled on engineering applications. For example, the 007 benchmark is modelled after an engineering application as shown in Figure 3-2.

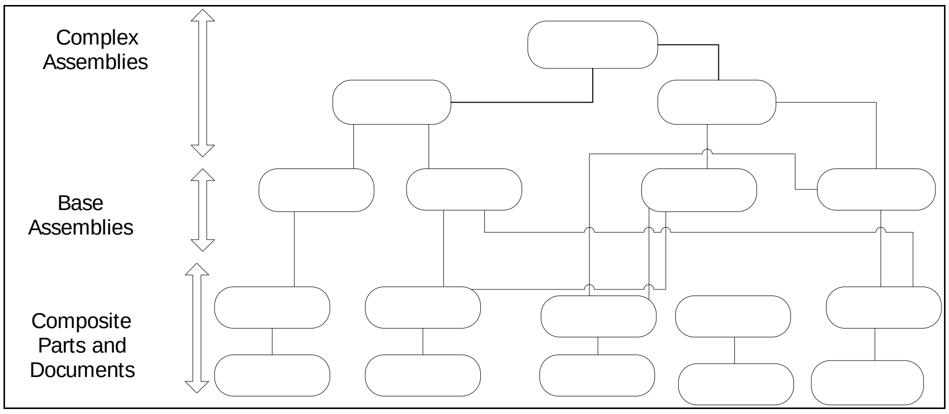


Figure 3-2 Structure of 007 benchmark modules (Based on figure 2 [CDN93])

The 007 schema has a separate table to store the information for each of the categories: complex assemblies, base assemblies, composite parts and documents. These categories and the type of data related to them is specific to engineering applications and so it is not likely that the exact tables would be required by the schemas for other application types.

The TPC suite aims to cover the whole business domain, using individual TPC benchmarks that focus on subsections of the domain. For example, the TPC-D benchmark has DSS schemas whereas the TPC-H benchmark has OLTP schemas [TRA02]. The FTR benchmark is another example, where the benchmark is designed after document systems and as such the schema is designed as a set of document partitions [DH91].

A slightly different approach to the design of the schema is found in benchmarks that are designed to evaluate a specific type of technology, often a new technology. Benchmarks aimed at testing a particular technology have a generic schema that includes the columns necessary to test unique functionality rather than being modelled on a particular application. Wisconsin [DEW93] and any of its derivatives [BD84, and BOT93] illustrate this well as their schemas do not relate to valid applications. BUCKY uses a different approach since its schema is modelled on a university application, as shown in Figure 3-3.

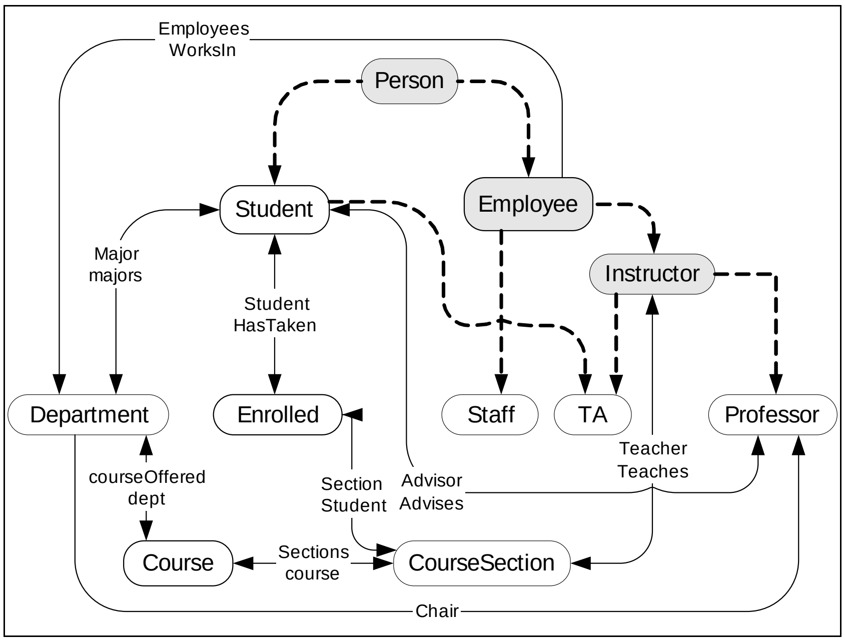


Figure 3-3 BUCKY benchmark schema structure (Based on figure 1 [CAR97])

The schema shown in Figure 3-3 is different from the two schemas previously seen in Figure 3-1, and 3-2 since it does not have to be modelled for a specific application type. The table names and data in BUCKY's schema could be completely different, whereas those of the SEQUOIA 2000 and 007 schemas cannot be altered as they are based solely on a type of application. The complex relationships between the tables is important with BUCKY's schemas as these relationships are used to test Object-oriented features, such as inheritance.

When a benchmark is designed to test new technology there is usually both a schema relevant to the new technology as well as a standard relational schema. Since the BUCKY benchmark is designed to evaluate ORDBMS, it has an Object-Relational (OR) schema which has an object for each item shown in Figure 3-3, and a relational schema which has all the items excluding those shaded grey. Rishe *et al's* [RIS00] semantic benchmark also has multiple schemas including a semantic schema, a sparse relational schema, and a compact relational schema.

3.4.2 Test Data

A well designed benchmark should provide test data as part of the benchmark suite. If test data is provided it must be decided what method should be implemented to generate the data and what data constitutes a representative sample (both size and format). Test data is normally either produced by synthetic means when needed [BD84, RIS00, and RUN02], or provided with the benchmark.

Synthetic data generation is typically used to produce test data when distinct but simple patterns are required. Examples of such distinct patterns include a normal Gaussian distribution (where data follows a bell shape) [RIS00], a Zipf distribution (where the data forms a straight line only when plotted with logarithmic scales on the axes) [RUN02], or when unique values are required. Examples of benchmarks that generate data synthetically are the Michigan benchmark [RUN02], where a template is used to generate descriptions, and the AS^{3}AP benchmark, where the data generator DBGen is provided with the benchmark.

It is much more difficult to produce a representative range of multimedia data by synthetic methods. The SEQUOIA 2000 [STO93] benchmark and the FTR benchmark [DH91] rather require that their multimedia data has to be downloaded with the benchmark. This method requires bandwidth to download the data and space to store the data, but allows a finer control on the test data.

Once the decision of how to provide the data is made, the format and amount of the data has to be determined. For the test data to be representative of a real workload, the amount of data must be considered in terms of the size of individual data items as well as the number of data items. Most benchmarks incorporate some form of scaling so that they cover a wider range of data sizes. This is achieved by providing multiple sizes of the databases. For example, the SEQUOIA 2000 benchmark provides a national, regional (x16) and global (x100) size database [STO93], the 007 benchmark has a small, medium, and large size database [CDN93], and the EDB benchmark has a standard size, a large database (x10), and a huge database (x100) [CAT93]. Other methods of scaling are seen in the FTR benchmark, which increases by 1 document partition for every 50 search transactions [DH91], and the Michigan benchmark, which increases the depth and fan-out values [RUN02]. Wisconsin measures scale-up by selecting a base hardware configuration and a base database size and proportionally increases both, for example doubling the size of the database as well as the number of processors [DEW93]. This method is very concise, but it is generally sufficient to use size-up, where the hardware configuration remains constant and the database size increases, to measure the scalability of most operations.

3.4.3 Workload

To define a benchmark workload, the type of queries, the resulting dataset, and the number of users must all be considered. Benchmark workloads could be representative of a particular application. For example, the workload of the 007 benchmark tests queries unique to engineering workloads, such as the speed of traversal, and the workload of the SEQUOIA 2000 benchmark tests 11 queries that are unique to Earth Science applications. Alternatively, a benchmark workload could be designed to test the most important functionality in a new type of application. For example, BUCKY has queries to test the additional feature found in ORDBMS that are not in RDBMS [CAR97]. The value of a workload that only tests a limited amount of functionality might be questioned, as its benefit is quite extensively restricted [CAR97].

A workload can be designed to test basic or composite queries. The first of these approaches is often used as the cost of performing composite queries can generally be calculated from the results of the simple queries. Runapongsa *et al* [RUN02] used this approach in the Michigan benchmark, where they only evaluated the basic queries that are performed when using an XML database. The aim of their benchmark was to assist database developers in understanding and evaluating alternative methods for implementing these basic operations. Another benchmark that evaluates basic queries is the Wisconsin benchmark, which has relational operations including selection, projection, deletion and modification, join, and aggregation. Boral *et al's* [BD84] benchmark focuses on the four basic query types that they feel are adequate to evaluate the performance of a system under a wide variety of workloads. In contrast, all the queries of the TPC suite are composite in order to simulate a set of queries usually found in a single transaction.

Most benchmarks simulate a single user environment, with the exception of Boral *et al.* [BD84] who defined a methodology to benchmark multi-user systems. They suggest that three factors influence the performance of transactions in a multi-user environment: multiprogramming level, degree of data sharing, and transaction mix. Multiprogramming level controls the number of copies of the program executing concurrently. This represents the number of users querying the database simultaneously, and can be up to a maximum of 16 in their benchmark. The degree of data sharing controls the data partition that is accessed. Three different degrees of data sharing are defined, 0%, 50% and 100%, where 0% means that separate partitions are accessed and 100% means the same partition is accessed by all programs. The transaction mix controls the queries executed, which is a selection from Query I to Query IV. The code is written so that it accepts these three parameters and tests the possible outcomes.

3.4.4 Performance Measurement

Performance measurements require decisions to be made about the performance metric, whether the final results are presented as multiple values or condensed into a single value, the manner in which the code is implemented to time the operations, and the unit that the results are in.

Performance evaluations can use a single performance metric or multiple performance metrics. If they use a single performance metric it is usually response time, also known as elapsed time. For example, response time is used in the Michigan benchmark [RIS00], the BEAST benchmark [GGD95], and the Wisconsin benchmark [DEW93]. Benchmarks that implement more than one performance metric include SEQUOIA 2000 [STO93] and the Full-Text document Retrieval (FTR) benchmark [DH91] which uses response time and price/performance, and the Transaction Processing Council (TPC) suite [TRA02] which uses these two as well as throughput.

Benchmarks can either present their results as multiple values, ranging from two to a few hundred, or they can use a formula to summarise their results into a single value. Examples of benchmarks that leave their results as multiple values include Michigan, which has 49 tests, and 007, which has 105 tests. Examples of benchmarks that summarise their results into a single value are BUCKY and EDB. Although SEQUOIA 2000 specifies that 11 numbers must be reported, it is also an example of a single-valued benchmark as these results are used to calculate a single value that measures the overall performance.

It must be decided what should be included in the timing. With response time it is usually easy to identify what needs to be included, but with throughput it is often more difficult. The problem is deciding exactly which bytes should be included. Musick and Critchlow [MC99] explain this dilemma:

> *"if the system reads 1Mb of useful information from a 3Mb interleaved array, but had to read the entire 3Mb to get it, was the total amount of data read 1Mb or 3Mb?"*

They use apparent throughput, which is much lower than the raw I/O throughput [MC99].

The unit in which the results are recorded, as well as their subsequent formatting must be decided on. For response time experiments, the results are most often recorded in seconds as seen in BUCKY, Wisconsin, and Michigan amongst others. Throughput results are usually measured in Mb/s, but there are variations of this, such as queries/unit or Mb/unit where the time unit can be minutes or even hours.

3.4.5 Programming Language

When designing a benchmark, the programming language is usually selected based on the functionality that it provides. For example, a database benchmark requires that the programming language offers functionality to connect to and query a database as well as to time the operations. Additional factors influencing the decision are the popularity and easy of use of the programming language at that time. Many of the benchmarks reviewed in this work were created around the early 1990s when C and C++ were the most popular languages. This is reflected in the fact that several of the benchmarks are written in these languages with embedded SQL [MC99]. For example, the TPC benchmark suite is coded in ANSI C and ANSI SQL2 [TRA02]. The EDB benchmark is also coded in C and embedded SQL, using a procedure called `time100()` that measures the time in seconds. A sample of the structure of this code is given in Figure 3-4.

```
starttime = time100();
for(j=0;j<mrepeats;j++)
{
      exec sql …;
}
endtime = time100();
printf( … , (endtime-starttime))/100);
```

Figure 3-4 Sample EDB benchmark C code

Another language often used is PERL. This language is used in the Wisconsin benchmark, which uses the PERL `DBI` module to communicate with the databases and execute SQL statements and the `Benchmark` module to time the operations [DEW93]. A sample of the structure of Wisconsin's code is shown in Figure 3-5.

```
$start_time = new Benchmark;
for($ti=0;$ti<$#table_names;$ti++)
{
      $sth=$dbh->do(“ …”);
}
$end_time = new Benchmark;
print”timestr(timediff($end_time,$start_time))”;
```

Figure 3-5 Sample Wisconsin benchmark PERL code

Specialised programming languages have also been implemented, such as MONET for the SEQUOIA 2000 Benchmark [STO93].

3.5 Problems with Existing Benchmarks

This section summarises the problems, for each aspect of the design, of the existing benchmarks to illustrate that none of them is directly suitable to evaluate a multimedia database. The main design areas that are reviewed are the schema, test data, workload, and programming language. The measurement of the performance is not reviewed because the requirements for this in existing benchmarks do not differ from that of a multimedia benchmark.

3.5.1 Schema Problems

The schemas of existing benchmarks usually have tables that consist of several attributes, where these attributes are mostly simple datatypes such as integers, strings or dates [BD84, BOT93, CAR97, CAT93, DEW93, RUN02, and RIS00]. These benchmarks are not designed for multimedia applications, and therefore their schemas do not contain any database tables that have columns that can hold multimedia data. Altering the schemas of existing benchmarks may produce a viable multimedia benchmark if they can be adapted to model a multimedia application without too much difficulty. Benchmarks where multimedia attributes exist include the 007 benchmark that has documents and associated “graphs of atomic parts” [CDN93], the FTR benchmark that has documents [DH91], SEQUOIA 2000 that has a raster attribute [STO93], and the TPC-W that has images [TRA02]. The problem is that none of these includes more than one multimedia datatype which makes them of limited use.

3.5.2 Test Data Problems

A multimedia benchmark must either use an appropriate method to synthetically generate multimedia data or provide the multimedia data with the benchmark. In the benchmarks reviewed, the data generators and the alternative method to synthetically generate data are unsuitable for multimedia data. It was not even possible to replace the sets of test data with suitable multimedia test data. The main problem is that it would be difficult to add the missing formats as neither the schemas nor the SQL code has been designed to handle such data. Only a few benchmarks already contain multimedia data, but none of them include all of the necessary multimedia datatypes. Of the benchmarks that do include multimedia data, the size of data is too small, with the exception of SEQUOIA 2000 and FTR benchmarks. For example, the MediaBench benchmark and the HERMES project have a limited variety of image and video formats [LPM97], while the TPC-W [TRA02] only has images. The test data of all the reviewed benchmarks is unsuitable and it cannot effectively be adapted.

3.5.3 Workload Problems

The existing benchmarks evaluate many different workloads, but none of them focuses on the unique requirements found in a multimedia workload. Some of them do test the same type of operations as those needed by multimedia databases, but the complex nature of multimedia data means that these operations have to be performed using special code as opposed to the standard code implemented in other benchmarks. For example, the insert operation for multimedia data not only needs an `INSERT` statement, but also an operation to upload or import the data into the database. Another problem is that none of the benchmarks test the advanced functionality, such as signature generation for images, essential in multimedia databases. These issues make the workload for a multimedia application significantly different, making other workloads unsuitable.

3.5.4 Programming Language

Many of the existing benchmarks use languages such as C, C++, and PERL. These languages appear to be suitable for a multimedia database benchmark as they have the functionality to connect to databases as well as time the operations. A programming language for a multimedia database should also, ideally, be easy to alter and port between systems, since multimedia databases implement their multimedia functionality in their own unique manner. Unfortunately, previously used languages are not as portable or easy to use as desired. It would also be beneficial if the programming language had built-in functionality to support multimedia data. An example of a programming language that is portable and also has functionality to support multimedia data is Java.

3.6 Required Benchmark

The required multimedia benchmark must have schemas, test data, and workloads designed for multimedia applications. The schemas must have at least one table that contains multimedia data, although ideally there should be a separate table for each multimedia datatype. Both the schema and the test data should include several types of multimedia data, especially images, audio, and video data, so that they produce a representative range of multimedia data. A multimedia workload must include typical queries that are found in multimedia applications to manipulate multimedia data. The benchmark must be able to evaluate the performance of a type of DBMS that supports multimedia data and its required functionality.

Table 3-1 summarises the main characteristics of each of the benchmarks so that they can be compared with the requirements of a multimedia benchmark.

Benchmark	**Schema/Test Data**	**Workload/Focus**	**DBMS**
Required	Multimedia tables and data	Multimedia Applications	ORDBMS
Wisconsin	No multimedia tables or data	Relational Features	RDBMS
BUCKY	No multimedia tables or data	OR Features	ORDBMS
AS3AP	No multimedia tables or data	Relational, OLTP	RDBMS
SPEC, MediaBench	No multimedia tables or data	Computer Architecture	None
TPC Suite (C, H, R and W)	Only TPC-W has images, no multimedia tables or data	Order-Entry, Business, OLAP, DSS and Web	RDBMS
EDB and 007	Only Documents in 007	OO Features, Engineering	OODBMS
BEAST	No multimedia tables or data	Active DBMS Workloads	ADBMS
FTR	Only Documents	Document Retrieval	FTR Systems
SEQUOIA	Only Images	GIS Applications	GIS Systems
Michigan	No multimedia tables or data	XML applications	XML DBMS

Table 3-1 Comparison of benchmarks

The first consideration is whether the benchmark includes multimedia data in its test data and schema, and the only benchmarks that do so are TPC-W, 007, FTR, and SEQUOIA 2000. Unfortunately, these benchmarks each have a single multimedia datatype, thus making the use as a multimedia database benchmark limited.

The second consideration is whether the benchmarks test workload that are similar to, or can be adapted to, a multimedia workload. The SEQUOIA 2000 benchmark has the workload that is the most similar to that of multimedia applications. This workload has a large amount of test data, including complex datatypes in the form of maps which could be replaced with images, and it also tests specialised search functionality. Unfortunately the nature of the queries in the workload differs significantly from those required by a multimedia application because they are very specific to GIS applications. The TPC suite, which includes some of the most widely used and popular benchmarks, are unfortunately also unsuitable as they are aimed at business orientated workloads consisting of specific characteristics that differ from that of multimedia applications. None of the benchmarks have a workload suitable for multimedia.

The third consideration is whether the DBMS that the benchmark evaluates supports multimedia data. Since RDBMS cannot support multimedia applications, any benchmark designed to test these databases is unsuitable. This eliminates benchmarks such as Wisconsin, all of the TPC suite, AS^{3}AP, and similar benchmarks. Although OODBMS benchmarks are feasible, they are usually designed for engineering applications that have workloads that differ significantly from multimedia applications. For example, the number of users, the queries executed, and the datatypes are different. A fairly well known ORDBMS benchmark is BUCKY, but unfortunately this benchmark has been designed to evaluate only a limited number of features. These features are not applicable to a multimedia workload, making this benchmark unsuitable.

3.7 Application of Benchmark

When implementing benchmarks, there are rules and restrictions that must be followed in order to ensure the accuracy of the results. An important issue that is seen in several of the existing benchmarks is the idea of *hot* and *cold* databases. A *cold* database is one where the cache is empty and the queries take significantly longer, while a *hot* database has a full cache and so produces faster, consistent results [CAT93, CDN93, and RIS00]. Examples of benchmarks that run their tests in both modes are, Rishe *et al.'s* [RIS00] Semantic benchmark, EDB, and 007. The general idea of *warming up* the database is to run the queries a number of times to fill the cache before the results for the "warm" database can be recorded [CAT93, CDN93, and RIS00].

The next important issue that needs to be considered is whether the benchmark should run in a networked environment. This greatly influences the results, and so it is surprising how few benchmarks specify this. Those that do specify it generally state that the benchmarks must be implemented in a networked environment. For example, the EDB has a client on one machine and the server on a different machine, and some of the TPC suite is especially designed to test networked environments.

3.8 Summary

Numerous benchmarks exist with which to measure the performance of databases: Wisconsin, Bucky, 007, AS^{3}AP, SPEC, MediaBench, TPC suite, Sun, EDB, Semantic, Michigan, SEQUOIA 2000, and FTR. Current benchmarks are categorised by their intended field of use: generic, business, engineering, advanced, and architecture. The objectives of each of these benchmarks were given and they were reviewed in terms of portability, simplicity, scalability, and relevancy. Most showed some if not all of these characteristics.

The design of each of these benchmarks was then analysed in terms of its schema, test data, workload, performance measurements, programming language and application. In these respects, no benchmark was found which is designed specifically to evaluate the performance of multimedia databases. The most significant difference between these existing benchmarks and the requirements of multimedia benchmarks is the workload.

Chapter 4 : Oracle DBMS

Chapter 2 surveyed multimedia databases and the tuning and evaluation of their performance. A comparison of the different database technologies revealed that ORDBMS are a viable choice to handle multimedia data. Oracle is an ORDBMS that has support for multimedia datatypes as well as their required functionality. This chapter provides an overview of Oracle, focusing mainly on the aspects that affect its performance. It describes the structure of Oracle, covering its architecture, components, and storage. A review of interMedia, which extends Oracle to provide the storage and management of multimedia data within the Oracle database, is presented. This chapter also briefly reports on research that has already been conducted on Oracle. Finally, a summary is provided of Oracle's performance, focusing mainly on multimedia data.

4.1 Introduction to Oracle

Oracle is a commercial ORDBMS that has all the features found in RDBMS as well as some of the features found in OODBMS. The RDBMS features include a well-defined relational structure, such as tables to store or access the data, predefined relational operations to manipulate the data, and integrity rules to control these operations. The inclusion of OODBMS features permits users to extend these features further, giving them, for example, the ability to define new object types. When new object types are defined to create new datatypes, both the structure of the data as well as the methods to manipulate it have to be specified. These datatypes can then be used within the relational model as if they were one of the built-in datatypes. With these capabilities, Oracle provides built-in functionality to support all multimedia datatypes. Multimedia support has only been available in the later versions including Oracle9*i* (release 1) version 9.0.1, which is used for the work done in this thesis.

4.2 Architecture of Oracle

A distributed architecture is often used to reduce the processing load of a single processor. This is achieved by splitting the required processing for a subset of tasks and allocating them to multiple processors. This improves both the performance and the capabilities of the whole system. Oracle utilises this technique in its architecture, and thus can be configured using 2-tier, multi-tier database architecture. These architectures can both be further distributed. Figure 4-1 illustrates the two architectures and the further distribution of the 2-Tier architecture.

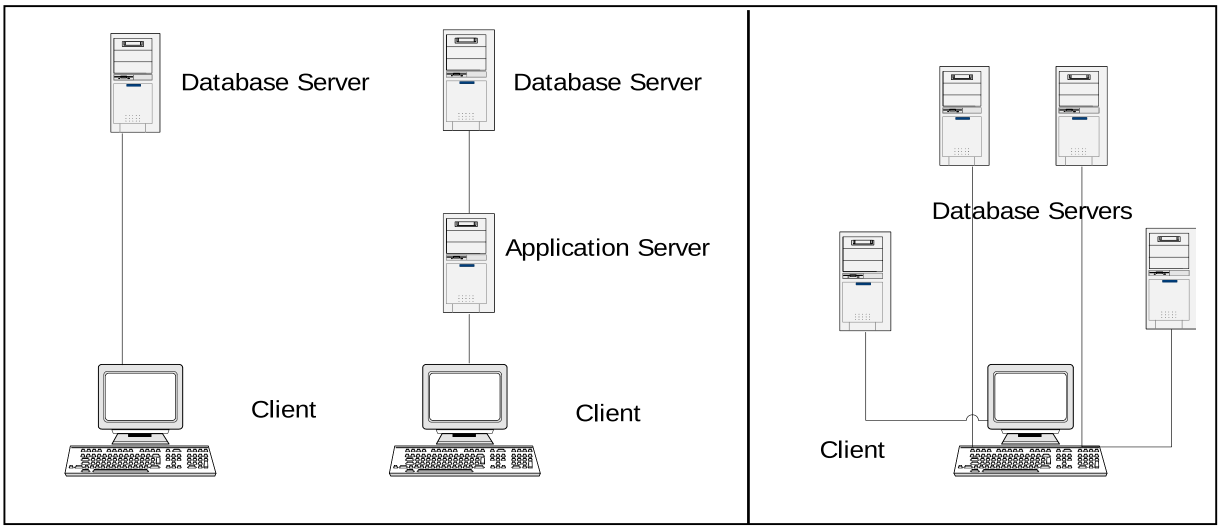


Figure 4-1 Types of architecture of Oracle (Based on figure 7-1[MCG01])

The 2-tier architecture is also known as client/server architecture as it divides the database system into a front-end client and a back-end server. Generally, this architecture has the user processes running on the front-end, while the server processes are running on different back-end machines. The tasks of the *user processes* are to request, process and present data that are managed by the server, but they are not responsible for the data access themselves. The task of the server is to run Oracle software to handle the concurrent, shared data access. The server receives the SQL and PL/SQL (Oracle's procedural SQL language) statements from the client and processes them in order to return the required data. A multi-tier architecture differs from the client/server architecture in that the server machine will have only a subset of the duties, although the client machine is very similar. There is at least one application server that acts as an interface between the client and the database server. Application servers perform most of the server-side processing such as validating credentials, connecting to databases, or executing user queries, reducing the processing load on the database server. The database server is now mostly used to store the data and control the operations performed by the application server.

The third setup shown in Figure 4-1 is a database distributed among multiple database servers, and even though the data is dispersed between the different servers, the database still appears to the user as a single logical database.

4.3 Oracle Components

4.3.1 Overall Structure

The main components of the Oracle 9i Server are the *server processes* and *Oracle processes* (for example log writer), the physical database files such as the *log files*, and the memory areas including the *System Global Area* (SGA) and the *Process Global Area* (PGA). This section only introduces these components, as they are discussed in greater detail in the following sections. Figure 4-2 illustrates how Oracle's components are related.

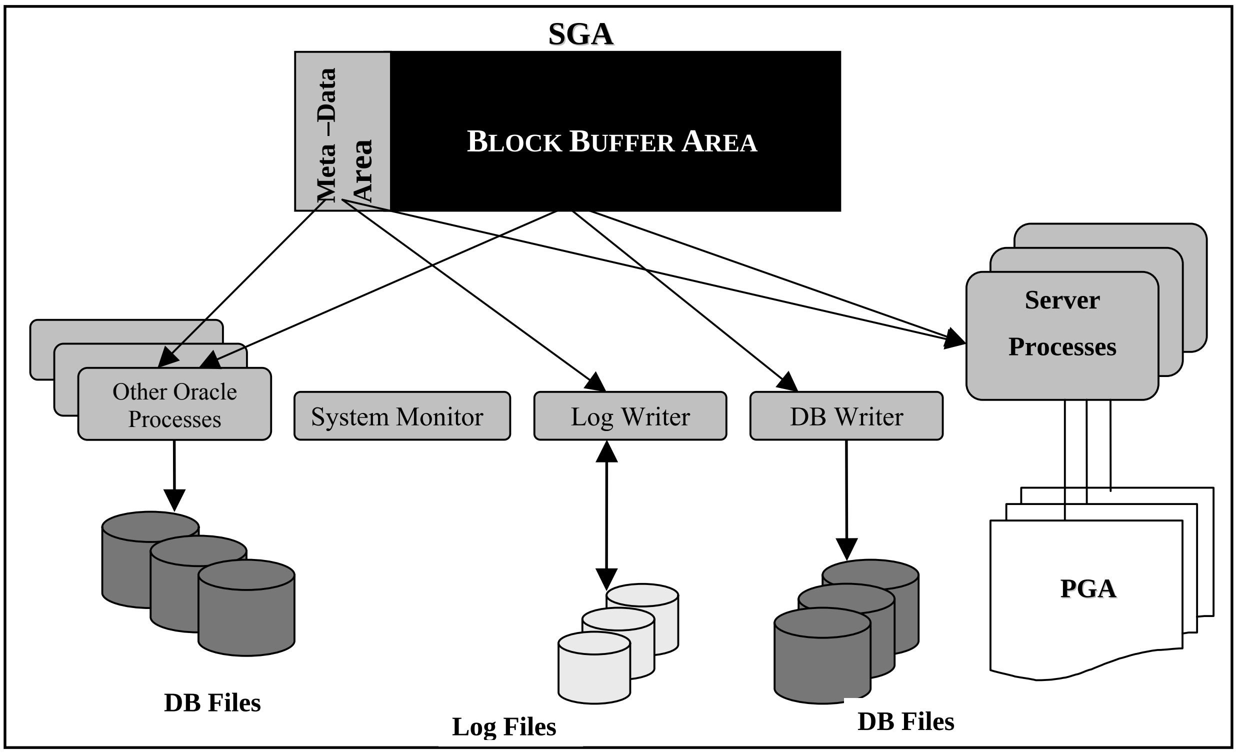


Figure 4-2 Main components of Oracle 9i server (Based on figure 1 [BGB98])

As just said, Oracle uses two types of processes, Oracle processes and server processes. The *Oracle processes* run in the background, performing the general database maintenance activities, while the *server processes* execute the database transactions, which makes them the more active of the two. Although the server processes account for most of the processing, the

Oracle processes perform important functions such as deadlock detection, system monitoring, writing to the database files, as well as writing the *redo logs* (records of all the database transactions).

The physical database files are persistent data structures that include the *tablespace files*, *control files*, *data files*, and *redo log files* (details in Appendix A.2). These files include the actual data of the database as well as database information such as how the physical storage is arranged. An example of this is the *redo log files* that keep a compressed log of committed transactions, which are used to restore the state of the database in the case of a failure.

The memory areas are the sections of the host system's physical memory that have been reserved for Oracle. Memory access is much faster than disk access, and as a result these regions store data for fast access. Assigning a larger size for the memory areas will increase the likelihood that the data is found in memory, which in turn reduces the number of disk accesses, thus improving the performance. However, it should not be too large as this wastes resources.

Oracle's memory model uses a hybrid structure that includes both *shared* and *private memory*. The *shared memory* region, known as the SGA, contains shared data as well as control structures, and is used mainly for communication and synchronisation. The *private memory* region is known as the PGA and is allocated by Oracle when a server process is created. A PGA memory region is assigned to each server process for its data, as shown in Figure 4-2. This region contains control and data information for that server process. In contrast, a single Oracle instance consists of the SGA and the relevant Oracle processes. The SGA is used to store code, such as database queries, that both the Oracle and server processes can access.

4.3.2 Oracle Processes

Oracle processes are background processes that start when an Oracle instance begins. These processes integrate functions such as I/O and monitoring of other processes, in order to provide better performance and reliability. They include the database writer, log writer, checkpointer, system monitor, procedure monitor, archiver, recoverer, dispatcher, lock manager server, job queue monitor, and queue monitor. The purpose of the *database writer* is to write the modified data blocks from the buffer cache in memory to the datafiles. Oracle's database writer uses a write-ahead logging algorithm which performs batch writes rather than writing to disk after each transaction commit statement. This means that the database writer only writes when the data in the SGA needs to be replaced, using the least recently used algorithm, which improves the performance and efficiency of the system. Oracle has a single background process, the *log writer,* that groups what are known as commit logs from independent transactions into a single disk write for more efficient use of the disk bandwidth. The commit logs become online redo log files, which the *archiver* can copy to archival storage. The function of the *system monitor* and *process monitor* are similar except that the system monitor is responsible for the recovery of a failed instance while the process monitor performs the recovery procedure for a failed user process. System monitor tasks include cleaning up temporary segments no longer in use as well as coalescing free extents in a dictionary-managed tablespace (see section 4.4.3 for information on extents). The *recoverer* is used to either commit complete transactions, or rollback pending transactions due to a network or system failure in a distributed environment. Oracle9i is the first release to include dynamic queuing which is controlled by the *job queue monitor* and is used for batch processing. The *dispatcher* and the *queue monitor* are optional processes, where the former is only used in a multithreading environment and the latter is only used with advanced queuing. The *lock manager server* process is only used in Oracle9*i* Real Application Clusters for inter-instance locking. An *Oracle instance* is the memory areas in addition to a set of Oracle processes.

4.3.3 Server Processes

Server processes control the communication between the user and Oracle. These server processes can vary depending on the number of user processes assigned to each. This is controlled by the configuration of Oracle, making the server processes either dedicated or shared, as shown in Figure 4-3.

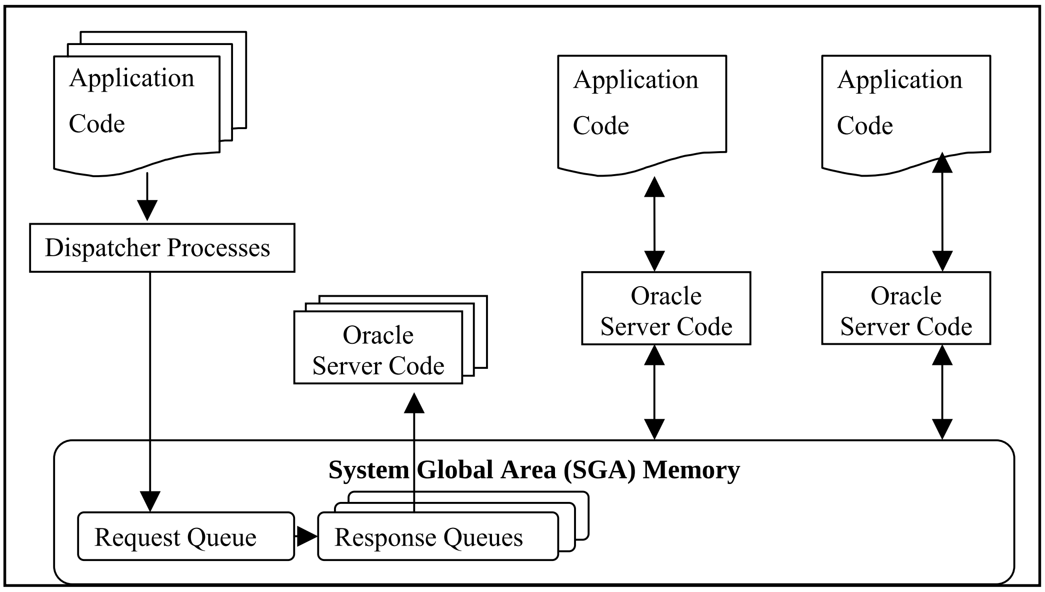


Figure 4-3 Dedicated and shared server processes (Based on figures 5-1 and 5-2 [BRF01])

A *dedicated server process* handles only the requests of a single user process (shown on the right of Figure 4-3), while a *shared server process* handles multiple user processes (shown on the left of Figure 4-3). Shared server processes minimise the number of server processes while maximising the use of available system resources. This means that when many users are accessing the database concurrently, the shared mode is better. Otherwise the dedicated mode is more suitable as certain operations, such as shutting down the database, cannot be performed in shared mode.

4.3.4 Physical Database Files

The physical database files consist of at least one datafile, only one control file, and at least two redo log files per database. *Datafiles* contain all the database data. Figure 4-4 illustrates the logical relationship between databases, tablespaces, and datafiles. DBFILE1, DBFILE2, and DBFILE3 are the names of the datafiles in Figure 4-4.

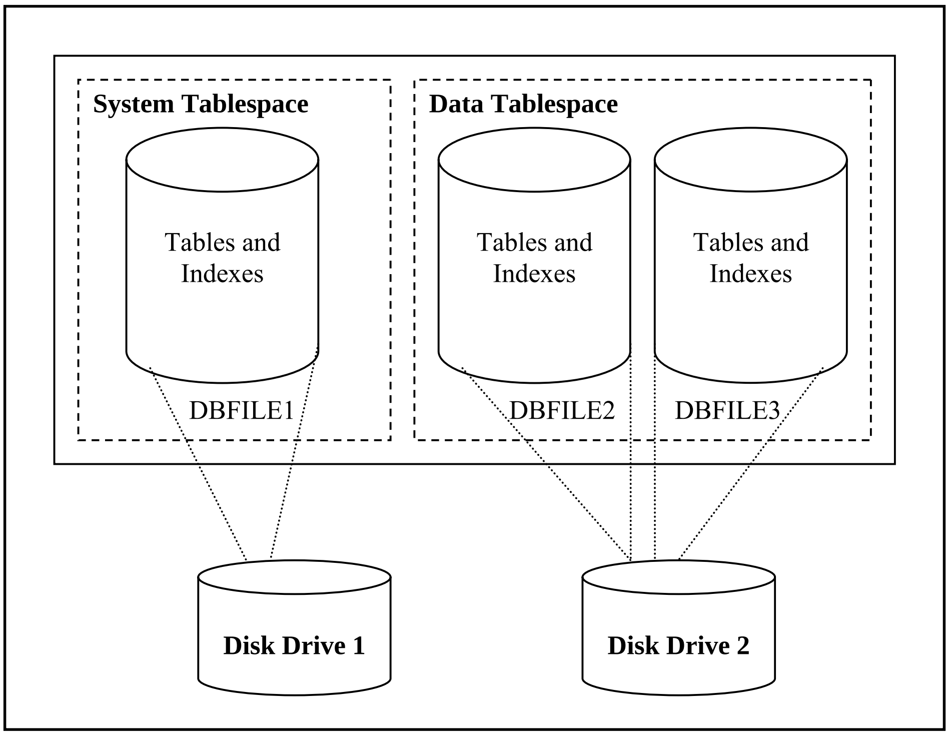


Figure 4-4 Relationships between databases, tablespaces, datafiles and tables (Based on figure 4-2 [MCG01])

It can be seen from the diagram that datafiles contain tables and indexes, collectively known as logical database structures. A datafile can be associated with only one database, while one or more of these datafiles form a logical unit of database storage called a tablespace. A *control file* contains information that specifies the physical structure of the database, including data such as the actual database name, the names and the locations of both its datafiles and redo log files, and a time stamp specifying when it was created. Control files are necessary to recover the database when there has been a failure, and they are also used to identify both the database and redo log files that need be opened when a database is started. The *redo log files* consist of many redo entries, also known as redo records, which record all the changes made to data to ensure work is never lost. If there is a failure that prevents the modified data from being permanently written to the datafiles, then the changes that were supposed to be made can still be obtained from the redo log. A set of redo log files is known as the database's redo log.

4.3.5 SGA Memory

The SGA memory consists of a larger section called the block buffer, covering approximately 80% of the SGA, and a smaller section called the meta-data area (Refer to Figure 4-2). The *block buffer* section contains the most recently used disk blocks and is the cache in main memory, whereas the *meta-data* section contains the directory information needed to access the block buffer as well as the space for other buffers and synchronisation data structures. The size of the SGA memory is controlled by a parameter known as the `SGA_MAX_SIZE,` which sets its upper limit. This size cannot be altered after the creation of the database, and thus its initial size must be calculated so that it is large enough to contain all the components of the SGA. These components comprise of the buffer cache, shared pool, large pool, log buffer, and the java pool. Figure 4-5 presents a sample configuration of the SGA, where each segment of the pie chart represents a different component of the SGA. (The values in this diagram are only used to illustrate how the SGA can be configured and are not the values used for the experiments, which are reported on later in the thesis.)

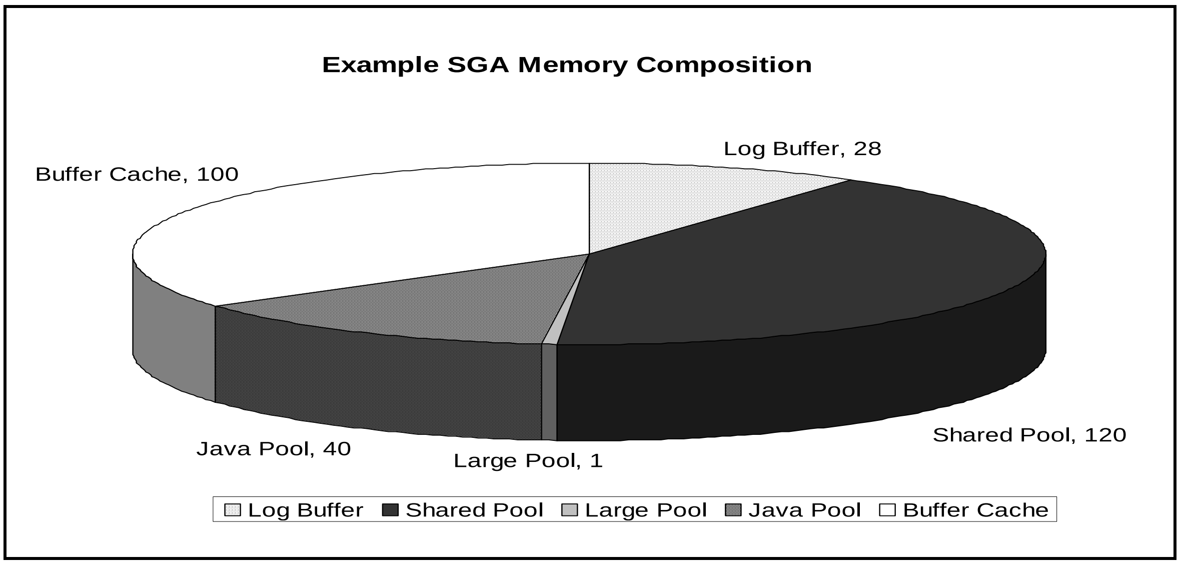


Figure 4-5 Main components of Oracle's SGA memory

From Oracle version 9 onwards, the sizes of these components can be dynamically increased or decreased during a session in units of memory called a granule as long as their total is less than the `SGA_MAX_SIZE`. A granule is 4Mb if the `SGA_MAX_SIZE` is less than 128Mb, otherwise it is 16Mb. The alteration of the size of these components is not instantaneous, thus there is a waiting period before the size of other components can be changed.

4.3.5.1 Buffer Cache

The *buffer cache* is the section of the SGA that is reserved for storing the data blocks that have been retrieved by the most recent queries. (These data blocks can either be 'modified', in that the value of the data has been changed, or 'unmodified', in that the value remains the same.) The buffer cache has a default pool for the standard block size, as well as one pool for each non-standard block size. (See section 4.4.2 for information on block size). There are also two additional pools for the default block size, known as the keep pool and the recycle pool. The *keep pool* is for frequently accessed data as it contains buffers that always stay in memory, while the *recycle pool* is for less frequently accessed data as the buffers are continually recycled.

Buffer Cache Replacement Strategy

When the buffer cache is full and more data needs to be retrieved, a buffer-replacement strategy needs to decide which data blocks are moved out of the buffer cache. To achieve this, the buffer cache in Oracle is divided into two lists, known as the write list and the Least Recently Used (LRU) list. The former holds *dirty buffers* that are the buffers that have been modified, but have not yet been written to disk. The latter holds a combination of free buffers, buffers currently being used, called *pinned buffers* by Oracle, and dirty buffers that have not yet been moved to the write list. Figure 4-6 illustrates the different scenarios that may occur when data is requested and the buffer cache is accessed.

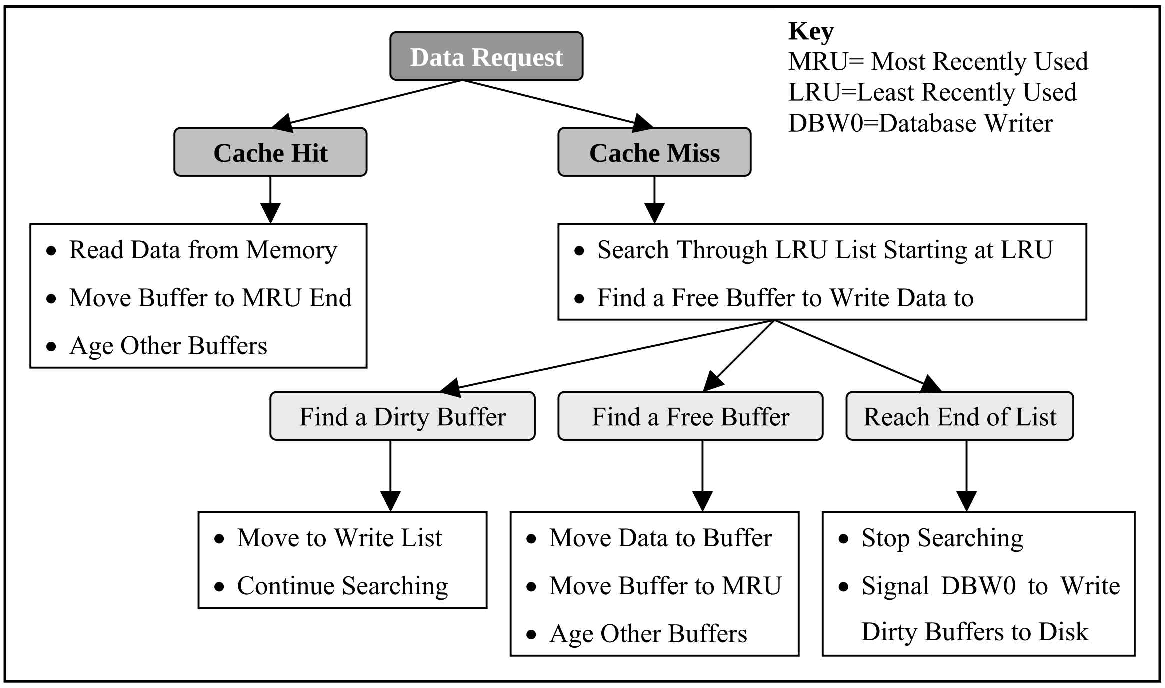
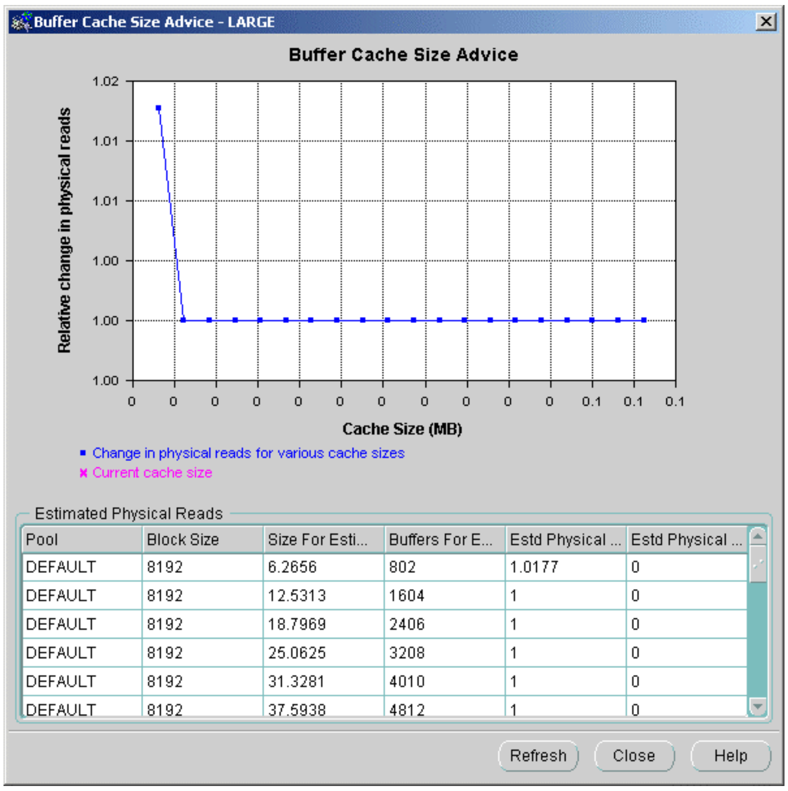


Figure 4-6 Possible actions when data is requested

The outcome depends on whether the data is already in the buffer cache, known as a cache hit, or whether it needs to be moved to the buffer cache*,* known as a cache miss. A *cache hit* is faster, as the data can be read straight from memory rather than from the disk. The buffer that is accessed is then moved to the Most Recently Used (MRU) end of the LRU list. This results in the other buffers aging, where they are moved toward the LRU end of the LRU list. Alternatively, if the required data is not in the buffer cache, a *cache miss*, then a free buffer needs to be located in order to move the data into the buffer cache. This is achieved by searching through the LRU list from the MRU end until a free buffer is found. Figure shows what actions are taken when the various situations occur.

4.3.5.1.1 Calculating the Performance of the Buffer Cache

Two methods can be used to aid in the understanding of the effect that the buffer cache has on the performance of the database. The first method uses Oracle's `V$DB_CACHE_ADVICE`, which must be explicitly set to `ON.` Oracle can then estimate how many physical reads would be required for twenty different potential cache sizes as illustrated in Figure 4-7.



Pool	Block Size	Size For Esti...	Buffers For E...	Estd Physical ...	Estd Physical ...
DEFAULT	8192	6.2656	802	1.0177	0
DEFAULT	8192	12.5313	1604	1	0
DEFAULT	8192	18.7969	2406	1	0
DEFAULT	8192	25.0625	3208	1	0
DEFAULT	8192	31.3281	4010	1	0
DEFAULT	8192	37.5938	4812	1	0

Figure 4-7 Screenshot of Oracle's buffer cache advice

(It can be seen from Figure 4-7 that Oracle's graph does not put the values of the cache sizes on the x axis as expected. These sizes can be found in the table under the 'Buffers for Estimate' column.) The use of the buffer cache advice does, however, introduce an overhead.

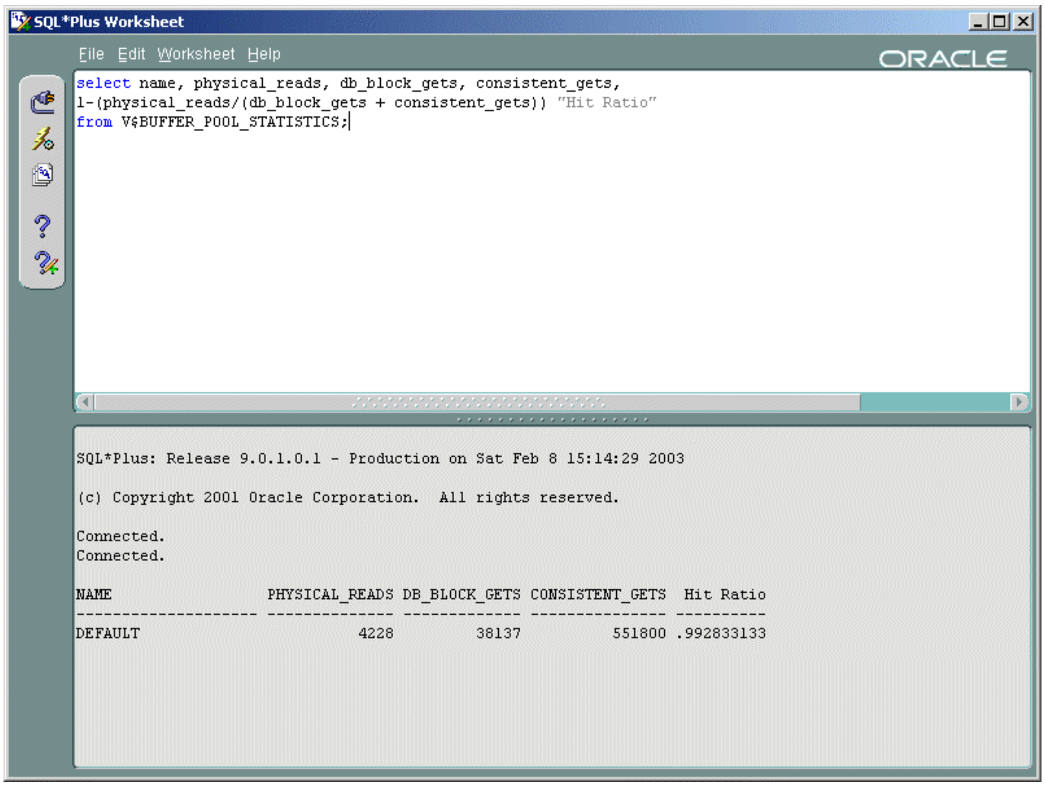
The second method uses Oracle's `V$SYSSTAT` tables directly, which store information to calculate the cache hit ratio. The formula for the hit ratio is:

Hit Ratio = 1-((physical reads – physical reads direct – physical reads direct (lob))/session logical reads)

Where: Session logical reads = total number of requests to access a block (whether this block is in memory or on the physical disk),
Physical reads = number of accesses to the physical disk,
Physical reads direct = number of accesses to data in memory excluding LOB data

Physical reads direct (lob) = number of accesses to the LOB data from memory.

To obtain the values for these variables, a table called `V$BUFFER_POOL_STATISTICS` is queried. Such a query can be executed through the SQL*Plus Worksheet, producing the results illustrated in Figure 4-8.



```
select name, physical_reads, db_block_gets, consistent_gets,
1-(physical_reads/(db_block_gets + consistent_gets)) "Hit Ratio"
from V$BUFFER_POOL_STATISTICS;
```

```
SQL*Plus: Release 9.0.1.0.1 - Production on Sat Feb 8 15:14:29 2003

(c) Copyright 2001 Oracle Corporation.  All rights reserved.

Connected.
Connected.

NAME                 PHYSICAL_READS DB_BLOCK_GETS CONSISTENT_GETS  Hit Ratio
-------------------- -------------- ------------- --------------- ----------
DEFAULT                        4228         38137          551800 .992833133
```

Figure 4-8 Screenshot of code and results for cache hit ratio

A higher cache hit ratio generally indicates that the buffer cache is the correct size, although it could be skewed by a dataset consisting of the same data being accessed repeatedly.

4.3.5.2 Shared Pool

The *shared pool* is the portion of the SGA that contains the shared memory components that all Oracle processes have access to. The components of the *shared memory* are the library cache, the dictionary cache, and the buffers for control structures and parallel execution messages. The *library cache* contains the parsed and compiled executable code, for example SQL or PL/SQL code that has recently been referenced. For the SQL code in particular there is a single SQL area that is shared, and as a result the code can be accessed by all the applications that issue the same statement. The *data dictionary* region holds the data dictionary. The data dictionary includes reference information about the database, such as its structure and users, and is accessed frequently. The remaining shared memory components differ from those previously mentioned in that their buffers contain structures that are specific to a particular instance configuration.

4.3.5.3 Large Pool

The *large pool* is the portion of the SGA that contains the session information for shared server systems, the message buffers for parallel execution, and the disk I/O buffers for the backup and restore processes. It is an optional area that is only relevant if there are parallel queries and is used where transactions interact with more than one database.

4.3.5.4 Redo Log Buffer

The *redo log buffer* portion of the SGA is used to store the logs of changes made to the database, known as the *redo entries*. These redo entries are then written from the redo log buffer to an active online redo log file on disk by the Log Writer (LGWR). The entries are used if database recovery is necessary. The size of the redo log buffer is set at configuration time and can be changed only by restarting the database. This cache is however less significant than the other caches as its size is small and it only represents a very small portion of the SGA.

4.3.5.5 Java Pool

The *java pool* is the portion of the SGA, which stores all session-specific Java code and data within the Java Virtual Machine (JVM). The java pool is used in different ways, depending on whether a dedicated or a shared server is running. In a dedicated server, the java pool stores only the shared part of each java class, such as code vectors and methods, and none of specific session data. In contrast, when using a shared server, the java pool holds both the shared part of as well as the session specific data with the result that the java pool must be larger.

4.3.6 PGA Memory

PGA memory is the region of memory, which contains data and control information for each server process. This includes the *private SQL area* that holds the run-time memory needed for executing SQL statements, and the *session memory* that holds session variables. A large portion of PGA memory is also dedicated to the *SQL work areas*, used for complex operations, such as sorts and hash-joins. As previously stated, PGA memory is private memory that is dedicated to a single server process when that process starts. This means that access is exclusive to the server process that owns it, and it is only written to and read from by Oracle code acting on behalf of that server process.

4.4 Oracle Storage

4.4.1 Storage Units

The logical storage of the data is controlled by Oracle and is divided into units known as data blocks, extents, and segments. The relationship of these units is shown in Figure 4-9.

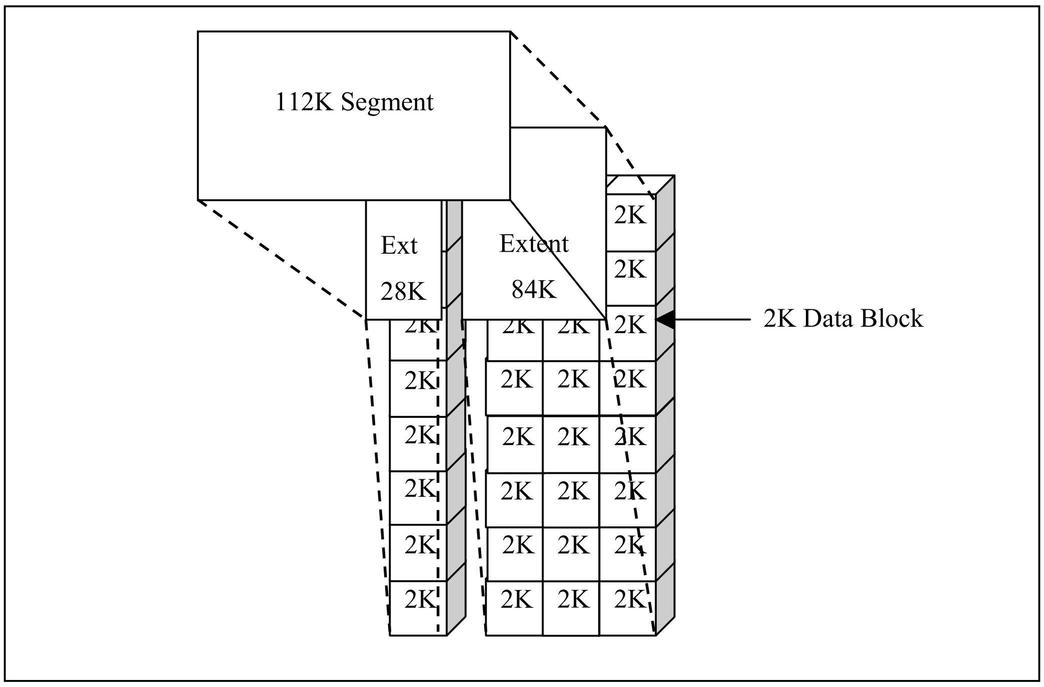


Figure 4-9 Relationship of segments, extents, and data blocks (Based on figure 3.1 [MCG01])

The smallest unit of space that Oracle can store its data in is a data block. These are also known as logical blocks, Oracle blocks, or pages. A number of adjacent data blocks can be grouped together in order to store a specific type of information, and is called an *extent*. The largest unit of logical database space is a set of extents that have been allocated for a specific data structure, called a *segment*.

4.4.2 Data Blocks

The Operating System stores its data in physical blocks measured in bytes, named data blocks. Unfortunately, Oracle stores and retrieves its data in units also known as *data blocks*. These differ from the operating system's block sizes in that their size is a multiple of the operating system's block size. For example if the size of the operating system's data block is 8K and its maximum is 32K then Oracle's data blocks can be 8K, 16K, 24K, or 32K in size. This size is set when the database is initially created and cannot be altered thereafter.

The format of all Oracle data blocks are the same, independent of the type of data that they store. Figure 4-10 outlines their format.

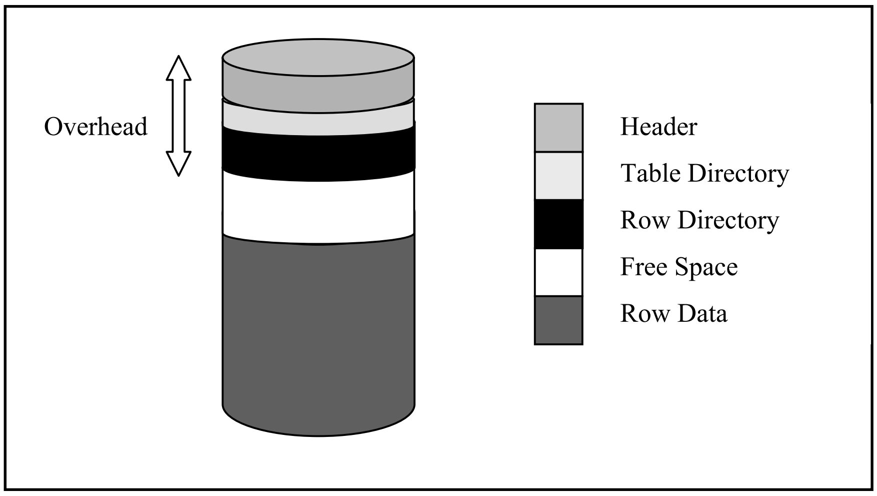


Figure 4-10 Format of data blocks (Based on figure 3.2 [MCG01])

Each data block contains a header, a table directory, a row directory, free space and row data. The header, table directory, and row directory contain information required by the data block, whereas the remaining sections are reserved for the actual data. Collectively the first three sections are known as overhead. The first section, the *header*, holds the general data block information, such as the address of the block. The other two sections hold information about the row data stored in the data block. For example, the *table directory* contains details such as the table that the row data belongs to, while the *row directory* contains details such as the location of the row data in the data block. The *row data* contains table or index data, while the *free space* is designated for inserting new rows as well as for additional space needed when existing rows are updated. The amount of free space in a data block is controlled by the variables `PCTFREE` and `PCTUSED`, as shown in Figure 4-11.

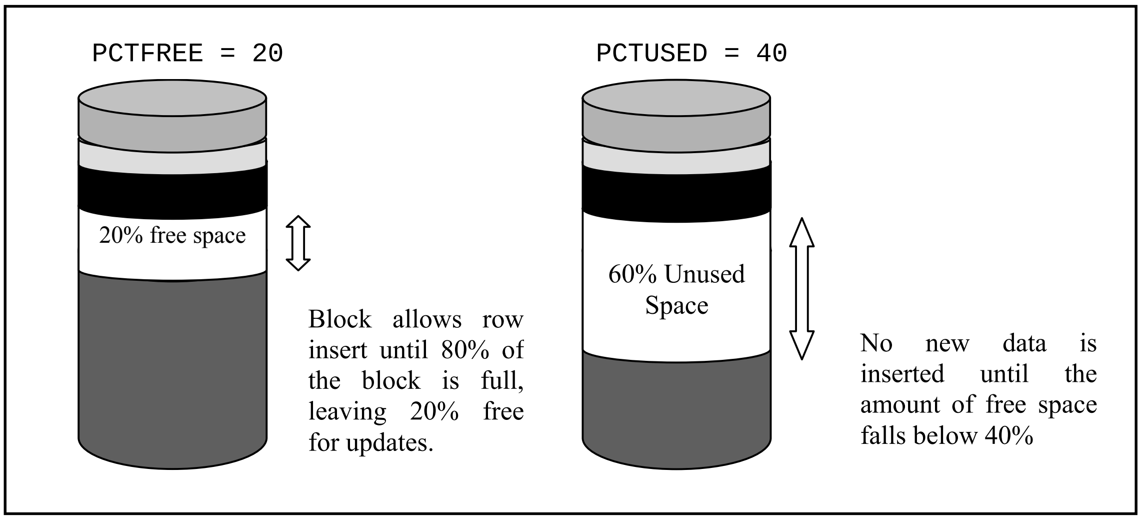

PCTFREE = 20
20% free space
Block allows row insert until 80% of the block is full, leaving 20% free for updates.
PCTUSED = 40
60% Unused Space
No new data is inserted until the amount of free space falls below 40%

Figure 4-11 Diagram illustrating data block storage control (Based on figures 3-3 and 3-4 [MCG01])

`PCTFREE` is the percentage of the block that is reserved for updates, and `PCTUSED` is the maximum percentage of the block after which Oracle will no longer consider that data block for insertion.

4.4.3 Extents

Extents are either initial extents or incremental extents. The *initial extent* is a number of data blocks that Oracle assigns when a table is created, and these are then reserved for the table's rows even when no data is inserted. When this initial extent becomes full, Oracle automatically allocates additional space, called the *incremental extent*. The incremental extent can either be the same size as the initial extent, or greater depending on the database configuration. These extents belong to a tablespace, which manages them either by the data dictionary or locally.

When using dictionary managed tablespaces, the space utilisation is controlled by the values for the storage parameters `INITIAL, NEXT,` and `PCTINCREASE.` These are specified when creating the table. The `INITIAL` parameter specifies the size in bytes of the initial extent, while the value of the `NEXT` parameter is the size in bytes of the incremental extent. `PCTINCREASE` is the percentage by which each incremental extent increases, in relation to the last incremental extent allocated for a segment. This means that if the value of `PCTINCREASE` is 0, then all incremental extents will be the same size. Two additional parameters can be set, called `MINEXTENTS` and `MAXEXTENTS`. The former specifies the number of extents that are allocated when the segment is created, while the latter specifies the maximum number of extents that can be allocated to a table. Locally managed tablespaces differ from dictionary managed tablespaces in that their space is automatically controlled by the system. The system achieves this by using bitmaps to track the used and free space.

Generally it is better to use locally managed tablespaces if there are a larger number of write operations such as `insert, update` or `delete` and dictionary managed tablespaces if the data is infrequently modified or read-only. Dictionary managed tablespaces allow the user to have more control as all the values are set by the user, but unfortunately there is a performance penalty due to the complex operations that it requires. The benefits of using locally managed tablespaces are that concurrency is improved, storage operations are faster, performance is improved, and space allocation is simplified. The disadvantage of using locally managed tablespaces is that they only support a few datatypes, such as `char` and `date.` This does not include multimedia datatypes, and therefore it is not possible to use it for a multimedia database.

4.4.4 Segments

There are four different types of segments in Oracle, known as data segments, index segments, rollback segments, and temporary segments. The first type of segment, a *data segment*, holds the data for a whole table if it is not partitioned or clustered; otherwise it holds one partition for a partitioned table or all of the tables in a cluster. This data segment is created when the table or cluster of tables are initially created. The second type of segment, an *index segment*, holds all of the data for a non-partitioned index or only the data for a single partition in a partitioned index. The third type of segment, the *rollback segment*, records the old values of data that has been changed by a transaction in order to provide a means to recover the database when necessary. Every database contains at least one rollback segment and they are only accessible by Oracle. Rollback segments contain several rollback entries that have details, such as the block information to identify where the changed data was, as well as the data before the transaction was executed. The last segment type, the *temporary segment*, is required as a temporary space for the transitional stages of SQL parsing and execution when processing queries. The first three types of segments are created in response to an explicit request of the user, while the temporary segment is automatically generated by Oracle when necessary.

In Oracle the free segment space can be managed either using bitmaps or freelists. A *bitmap* describes the status of each data block, based on the amount of the available space in it in relation to the total amount of space. This status then varies under the control of Oracle as the available space is changed. Alternatively *freelists* can be used, where a list of the data blocks that have free space available for inserting data is kept. This type of segment management is not automatic controlled, with the disadvantage that various parameters have to be set before it can be used. Freelists have the advantage over bitmaps in that they can be implemented for both locally and dictionary managed tablespaces, whereas bitmaps can only be used for dictionary managed tablespaces.

4.5 Oracle Multimedia

4.5.1 InterMedia

Oracle *Inter*Media is a standard feature that extends Oracle9i to provide internal storage and scalable management of multimedia data. Multimedia data, such as images, audio, video, and geographical location information, can therefore be managed directly by Oracle in an integrated manner with other standard data. Among the services that *inter*Media provides are, metadata management, support for popular formats, and content-based searching. The integration of *inter*Media with the architecture of Oracle, using a 3-tier architecture, is shown in Figure 4-12.

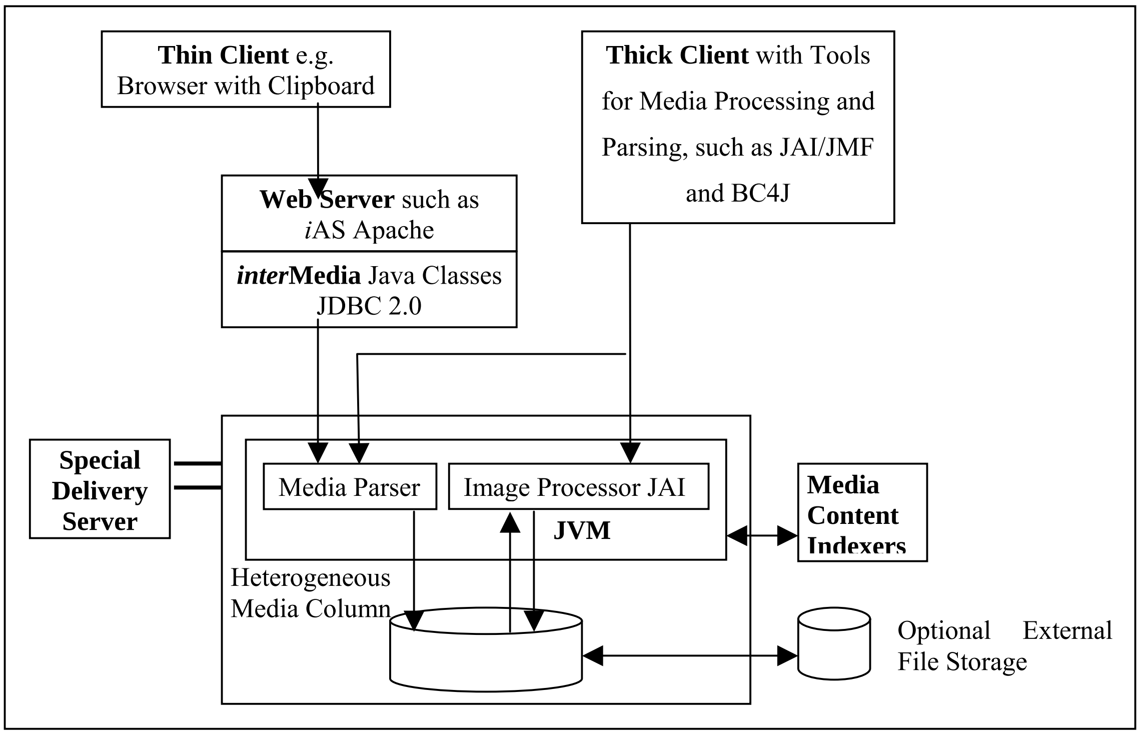

Thin Client e.g. Browser with Clipboard
Thick Client with Tools for Media Processing and Parsing, such as JAI/JMF and BC4J
Web Server such as *i*AS Apache
***inter*Media** Java Classes JDBC 2.0
Special Delivery Server
Media Parser
Image Processor JAI
JVM
Media Content Indexers
Heterogeneous Media Column
Optional External File Storage

Figure 4-12 Integration of InterMedia with architecture of Oracle (Based on figure 1-1 [CHI01])

The multimedia data can either be stored in the tables with the standard data, called heterogeneous media columns in the diagram, or in external file storage. This storage is connected to Oracle9*i's* Java Virtual Machine (JVM) that provides a server-side media parser as well as an image processor. The media parser, through either an object-oriented or a relational interface, parses formats and metadata, as well as registers new formats and extensions. The image processor performs image processing, such as creating thumbnail-sized images or image indexing and matching. *Inter*Media has upgraded its image processor to use Java™ Advanced Imaging (JAI) as it offers a rich, open, platform portable imaging package. The advantage of this is more supported file formats and processing operations, as well as additional supported file COmpression and DECompression schemes (CODECs). The media parser and image processor then connect to the Internet Application Server (*i*AS) or an alternative web server that provides access to *inter*Media through special *inter*Media Java classes. These Java classes enable applications on any tier to access, manipulate, and modify multimedia data stored in the database. This results in applications easily selecting and operating on result sets that include both traditional relational data and *inter*Media media objects. These applications can be run on a thin client with a browser-based interface and a clipboard provided by *inter*Media on which multimedia data is manipulated. Alternatively, a thick client can be implemented, which provides various methods for developing applications. Among these are media processing and parsing through JAI and the Java Media Framework (JMF), and building scalable, multi-tier applications with Business Components for Java (BC4J) that is JDeveloper's programming framework.

4.5.2 Oracle’s Multimedia Datatypes

Oracle is an object relational DBMS with additional capabilities provided by *inter*Media and so provides different datatypes for multimedia data. These include the support of Binary Large OBjects (BLOB), large objects located on the file system called BFILEs, URLs containing multimedia data stored on any HTTP server, and *inter*Media object types. Table 4-1 gives a comparison of the multimedia datatypes provided by Oracle.

	BFILE	**URL**	**BLOB**	***inter*Media**
Storage Location	Local File System	HTTP Server	In Database	In Database
Transactional Control	None	None	Full Control	Full
Data Access	Read Only	Read Only	Read/Write	Read/Write
Standard Functionality	Insert-Pointer, Select, Delete	Insert-Pointer, Select, Delete	Insert, Select, Update, Delete	Insert, Select, Update, Delete
Multimedia Data Specific Functionality	None. Pointer to multimedia data location	None. Pointer to multimedia data location	None. Stored as unstructured object	Yes e.g. Content-based retrieval, scaling images, streaming video

Table 4-1 Multimedia datatypes provided by Oracle

The table shows that BFILEs and the URL datatypes do not actually store the data in the database, with the result that the database has limited control over it and no transactional control is an example of this. These datatypes may be a suitable mechanism for situations where there is a large repository of multimedia data that already exists on a file system, but this solution is not advisable as the limited control may lead to corruption of data or data redundancy. Ideally, it is better to store the data within the database using the BLOB or *inter*Media datatypes. The functionality provided by the BLOB datatype is still limited since the multimedia data is only stored as unstructured binary object. There is therefore no differentiation between the different multimedia types with the result that a method specific to one type of multimedia data, such as scaling images, cannot be provided. *Inter*Media's object types are the most advanced multimedia datatypes, with a different datatype for each kind of multimedia data. These object types are known as *ORDDoc* for heterogeneous data, *ORDImage* for digitised image data, *ORDAudio* for digitized audio data, *ORDVideo* for digitised video data and *ORDSource* for any of these four types. An additional method is available in Oracle for storing video and audio data on specialised media servers, such as Oracle video server, for streaming purposes. This data can be streamed using an Oracle *inter*Media plug-in that supports the streaming server, and then delivered to the client, which uses the browser-supported streaming player, for playback. In addition to these datatypes that *Inter*Media offers for multimedia data, it also provides the methods necessary to manipulate this multimedia data.

4.5.3 Specialised Multimedia Functionality

Standard database methods, as well as more advanced functionality are provided for multimedia data by *Inter*Media. Inserting and selecting are examples of standard functionality, whereas the extraction and displaying of metadata (information about the data) are more advanced functionality. *Inter*Media extracts the metadata that contains general information, for example the data format, and datatype specific information, for example the frame rate of video data. This data is then stored in the database under Oracle's control and can be used to correctly present the multimedia information by either the user or an application program. Other advanced functionality provided is streaming capabilities for audio and video data, and several methods for image data, including content-based retrieval, generation of thumbnails, scaling, and cropping. Of these, content-based retrieval is fairly new and more complex than the other functions and thus it is briefly discussed here.

To perform content-based retrieval the information stored in an image must be processed so that the image can be abstracted in terms of its visual content. This abstraction can then be used to compare or match images by appearance. Oracle analysing every image inserted into a database and then produces the abstraction by generating a compact representation of the image's content, called a feature vector or signature. The signature contains information about the colour, texture, shape and position of shapes within the image, within 3000 to 4000 bytes in general. After the signatures have been created, content-based retrieval is done by first presenting a comparison image to *inter*Media. *Inter*Media compares this image's signature with the signatures of selected images in the database and then ranks the images from the closest match down until a threshold is reached. Ranking is done based on specified weightings for the criteria, and the threshold is the value below which images are regarded as too dissimilar and therefore will be excluded from the result set.

4.5.4 Multimedia Storage and Retrieval

The storage of multimedia data in the database tablespaces is in a manner that optimises space utilisation and makes access efficient. From the access of the multimedia data, it appears to be fully integrated with the other standard data but this is not the case. There is an option when creating a table with multimedia data to specify whether the data is stored *in line* (with other data) or *out of line* (separately). Independent of this specification, Oracle however only stores data that is less than 4Kb in line with other row data, and automatically stores larger multimedia data out of line. A locator is then generated to point to the location of the actual data and is stored in the row, while the actual data, which can be up to 4Gbs, is stored on a database page that is separate from the other data. This process happens transparently.

Multimedia data is larger than other datatypes and thus its retrieval typically requires a greater number of disk accesses and more time. To reduce this time various methods can be implemented, among them are using a bigger block size, increasing the size of the buffer cache or setting the value for the parameter `CHUNK` size as large as possible. The first two methods have been mentioned previously in 4.4.2 and 4.3.5.1 respectively. The `CHUNK` size is the number of blocks retrieved simultaneously and is specific to the retrieval of multimedia data. It can be used to force Oracle to retrieve multiple data blocks together, so reducing the amount of disk I/O.

4.6 Related Oracle Research

4.6.1 General Oracle Performance Research

Oracle Corporation has done a substantial amount of work on evaluating the performance of their products, including detailed documentation on performance tuning. Among the available white papers are comparisons with other commercial DBMS, such as Microsoft’s SQL Server 2000 and IBM’s DB2 [OTN02]. White papers are also available on the performance in DSS (Decision Support System) environments and for E-Business applications. A white paper on the optimal configuration of physical storage using a Stripe and Mirror method [LO01] is one such example. In this white paper, Loaiza [LO01] explains that due to the complexity of DSS workloads, it is difficult to configure the storage efficiently, but Oracle can improve the performance of load operations by using *direct Input/Output (I/O)* and *asynchronous read ahead*. Direct I/O is where the buffer cache is bypassed and the disk is written to directly, and asynchronous read ahead or double buffering is where multiple asynchronous operations are performed in one I/O operation. The goal of these is to stop operations from being I/O bound by producing the maximum disk throughput and usually then making them CPU bound. Other areas that the white papers focus on are *memory management, segment management*, and *direct I/O management*. *Memory management* investigates the effect that the dynamic SGA and buffer cache management have on performance, *segment management* looks at the effect of freelist*s* versus bitmaps, and direct *I/O management* investigates special I/O features. An example of the latter is data block pre-fetching, which is an internal optimisation that improves the response time by delaying the reading of a table until a number of satisfying ROW IDs have been accumulated. Oracle then prepares the buffer cache by issuing simultaneous I/Os for blocks containing all those ROW IDs. Independent work has been done by Kreines and Laskey [KL99] on tuning an Oracle database. They emphasise that the successful tuning of it requires the following to be addressed:

- Hardware and operating system performance
- Oracle instance performance
- Individual transaction (SQL) performance

Operations should ideally perform at the speed that the physical hardware permits, called the device speed, but this does not often happen and so I/O determines the upper threshold of the load performance.

4.6.2 Binary Large Object Performance Research

Shapiro and Miller [SM99] investigated the performance of Oracle when using Binary Large Objects to make recommendations on how to save storage space and processing time. Binary Large Objects (BLOBs) is a datatype for storing multimedia data, but in an unstructured manner (See section 4.5.2). The three storage systems that were investigated are a database with data stored externally (URL), a database with data stored internally (BLOB datatype), or a file system, which was UNIX in this case. These storage systems were then tested for three different database sizes (0.5 GB, 1.5 GB, and 3 GB), and two different BLOB sizes (0.5 GB, and 5 GB). The two workloads that were tested are classified as "select-dominant" and "Insert/Update-dominant" and their results are shown in Table 4-2.

	File System	**Database - External**	**Database - Internal**
Select – Dominant Workload	Larger Overhead	Small Overhead	Small Overhead
Insert/Update Workload	Smaller Overhead	Medium Overhead	Large Overhead

Table 4-2 Results of BLOB experiments (Results taken from [SM99])

Shapiro and Miller [SM99] found that for a "Select-dominant" workload similar times were produced for internal and external datatypes but the overhead for the file system was larger than the overhead for these database storage systems. In contrast, for the "Insert/Update-dominant" workload they found that the database overhead was larger than the file systems, and internal database storage took the longest, by a factor of approximately seven. They suggest that the long time is as a result of the large number of I/O operations that Oracle requires for rollback and log information as well as the processing of multiple migrations and chaining rows.

Three areas of tuning were also examined, namely design tuning, application tuning and memory and I/O tuning. When looking at the areas of tuning they managed to increase the performance of both workloads by 3% by *pinning* all the SQL statements, which ensures that the statements stay in the library cache. For the memory and I/O tuning it was found that by altering the `DATA_BLOCK_SIZE` the response time improved by 12% for the "Select-dominant" workload and 30% for the "Insert/Update-dominant" workload. The response time was dependant on whether the size of the `DB_BLOCK_BUFFERs` was increased or decreased, where an increase produced an improvement. They conclude that the `DB_BLOCK_BUFFERS` should be at least equal to 150% of the size of the largest data in the database. Overall, with tuning the performance of a database, an improvement in the performance of 60% was achieved. Finally, they found that an increase in either the database size or the average object size significantly decreased the performance.

4.6.3 Multimedia Oracle Performance Research

Results from the benchmarking of multimedia applications have been made available by Oracle [OIM02]. These test the loading and retrieving of multimedia data, but unfortunately do not include updating, modifying, or deleting of multimedia data. In the loading experiments, the performance of a workload with between 1 and 8 large (250MB) videos was investigated. Table 4-3 presents the results obtained for the response time, throughput and % CPU utilisation for these experiments.

# of Streams	Elapsed Time (Sec)	Throughput (MB/Sec)	% CPU Utilisation
1	24.8	10.1	8
2	63.6	7.9	12
4	79.3	12.6	13
8	177.7	11.3	14

Table 4-3 Database load performance of 250MB video files (Table from [OIM02])

They found that the bottleneck was the I/O bandwidth of the I/O controller as a maximum throughput (12.6 MB/sec) was obtained when a parallel load stream was used. A second workload that represented a web environment consisting of 240,000 distinctive objects, ranging from 4Kb images to 2Mb audio files, was tested. Full results for these tests and the details of the distribution of data were not given in this white paper. The results that were presented were for the throughput of the load experiment, which was found to be 7.5Mb/s, and the retrieval experiment, which was approximately 7Mb/s. This shows impressive performance, however the performance enhancements attributable mainly to the specialised hardware, including RAID, 4GB of RAM and 4 processors, must be considered when analysing these results.

4.6.4 Oracle Memory Research

Dageville and Zait [DZ02] completed work that focused on evaluating the performance of Oracle9i with different configurations for the SQL memory management. They used DSS as well as OLAP workloads, and focused on SQL Memory, which consists mainly of the Process Global Area (PGA) memory. Dageville and Zait define two important thresholds known as the *one-pass size* and the *cache size*. The former is the size of the work area needed to process the input data, whereas the latter is the total size of the work area. Dageville and Zait believe that the amount of memory greatly affects the performance and thus the work area should be large enough to accommodate both the input data as well as the auxiliary memory structures. Problems occur as the amount of memory in a system is limited, and furthermore, the workload is not easy to predict, making tuning of the database difficult. They demonstrate that the system adapts the configuration of the PGA memory to set the optimal values in response to the workload.

4.7 Oracle Performance Summary

Past research on the performance of Oracle, with respect to multimedia data, has shown that the CPU utilisation is not significant as it is generally below 15%. It was also found that the maximum throughput was achieved with parallel load streams, suggesting that the I/O bandwidth was the bottleneck. This is supported by the findings from work conducted by Shapiro and Miller [SM99]. Although they researched BLOBs rather than multimedia data stored using *inter*Media datatypes, they found that performing I/O tuning improved the response time by 12% for “select-dominant” workloads and 30% for “Insert/Update-dominant” workloads. I/O tuning is done by altering the configuration of Oracle’s storage, more specifically the parameter `DB_BLOCK_SIZE` [CG01]. This parameter controls the smallest unit of space, and has to be a multiple of the size of the operating system’s data block (see section 4.4.2). Another significant finding from the work done by Shapiro and Miller was that memory tuning also greatly improved the performance of the database. Memory tuning was the focus of Dageville and Zait’s research, where they studied the effect of the performance in relation to the size of the PGA memory. PGA memory is the private memory that is used for complex operations, such as sorts and hash-joins, which multimedia data does not generally require. Although the size of PGA is thus unimportant for multimedia database, the results from studying the PGA can still be used to help understand the affects of the size of the SGA, which is important. The SGA is the shared memory region that consists of the buffer cache, Java pool, large pool, shared pool, and the log buffer [CG01]. Of these components the buffer cache is the most influential on the database’s performance, especially when dealing with large data volumes and read intensive workloads. In summary, a performance evaluation of multimedia databases should focus on memory and I/O tuning.

4.8 Summary

Oracle is a commercial ORDBMS that has built-in functionality to handle multimedia data. Oracle has opted for a distributed configuration that uses either a client/server architecture or a 3-tier architecture, both of which can be further distributed. The components of Oracle consist of processes, memory regions, and physical database files. The processes are either Oracle processes or server process, while the memory regions are either private known as PGA or shared known as SGA. The physical database files include datafiles, control files, and redo log files. Oracle’s storage is divided into units with the smallest being data blocks, then extents, and the largest being segments. Oracle has different datatypes to support multimedia data such as BFILE (externally), URL (externally), BLOBs, and a more advanced method through *inter*Media. *Inter*Media is a standard feature that extends Oracle to provide the support for the datatypes and functionality for multimedia data. Research has been conducted on the performance of Oracle that focuses on standard datatypes, BLOBs, multimedia data, and the PGA memory region. Although this research acts as a good basis for understanding the performance of Oracle, further research needs to be conducted on the performance when handling multimedia data.

Chapter 5 : Design of MORD

Chapter 2 reviewed work in the area of multimedia database tuning, while chapter 3 reviewed work on benchmarks, both relating to this thesis. Chapter 4 provided an overview of Oracle. This chapter describes the design of a new database benchmark (MORD) that measures the performance of basic functionality in multimedia ORDBMS. MORD's objectives include portability, simplicity, scalability, and relevancy. MORD's components include the database schema, test data, and workload. The implementation decisions regarding the performance measurements and programming language to be used are also discussed in this chapter. Finally, a set of instructions is specified that should be followed to ensure that MORD is applied correctly to a test multimedia database system.

5.1 Design of a New Benchmark

The survey presented in chapter 2 revealed that multimedia databases have unique requirements not found in other databases. A review of the available database technologies indicated that ORDBMS, such as Oracle, are the most suitable systems to handle multimedia data. Performance is of the greatest importance with databases, and especially with multimedia database where large data volumes and complex operations are common. Tuning needs to be carried out, and is most easily achieved using an appropriate benchmark. Various benchmarks were reviewed in chapter 3, revealing that none of these benchmarks are designed specifically to evaluate the performance of multimedia databases. The most obvious shortcoming is that both the test data and the database schemas do not include multimedia data. It is complex to adapt these benchmarks, as their whole structure is founded on their schemas and test data. A new benchmark needs to be designed specifically to evaluate the performance of multimedia databases. Since Oracle is an ORDBMS that adequately handles multimedia data and its additional functionality, it is used as the target database for the benchmark.

Once designed, MORD's functions must be tested for correctness by performing some tests as discussed in later chapters. These tests are based on the knowledge gained about database tuning in chapter 2, as well as the specific Oracle implementation details given in chapter 4.

5.2 Objectives of MORD

The benchmark designed for this thesis, called MORD, must be able to evaluate the performance of basic functionality in multimedia databases in multimedia databases. It is therefore a domain-specific benchmark and must satisfy the criteria required of these benchmarks, which are portability, simplicity, scalability, and relevancy as described in section 3.3.

To be portable, a benchmark has to be designed to be applicable to a range of systems. Given the nature of multimedia databases, for which no standard implementation is available yet, portability is difficult, if not impossible to satisfy. In particular, each multimedia database has its unique datatypes and methods for handling multimedia data, so designing the benchmark for one type of multimedia database makes it unsuitable for an alternative one. With this in mind, it is not possible to design MORD to be a fully portable multimedia database benchmark until multimedia databases have been standardised. Rather the objective when designing this benchmark was to make it as generic as possible. Important design issues that influence this are the way that the code is written. For example, it should be modular so that functions can easily be replaced. Coding language is also important; making Java a suitable language to use as it is more portable than other coding languages.

To be simple, a benchmark must be easy to use and extend. These are satisfied by MORD being designed to be simple both in it ease of use as well as its comprehensibility, which makes it easy to adapt.

To be scalable, a benchmark must evaluate small and large databases. MORD is designed to be scalable by having three databases each of different sizes (small, medium, and large) -

adding more data can create additional sizes and the amount of data manipulated for each query can be altered.

Finally, to be relevant, a benchmark must be designed to evaluate standard operations found in a "real" application. MORD is relevant in that it is designed to test the basic functionality, such as inserting and deleting, which are essential in all multimedia database applications. Unfortunately MORD does not fare so well in terms of relevancy in that it does not evaluate more advanced functionality, such as streaming videos, which would also be found in multimedia database applications.

MORD was designed to assist with the configuration of multimedia database and thus the test database must be able to support a variety of datatypes and functions. For example, Figure 5-1 shows the image, audio and video data that is produced in the Computer Science Department at Rhodes University.

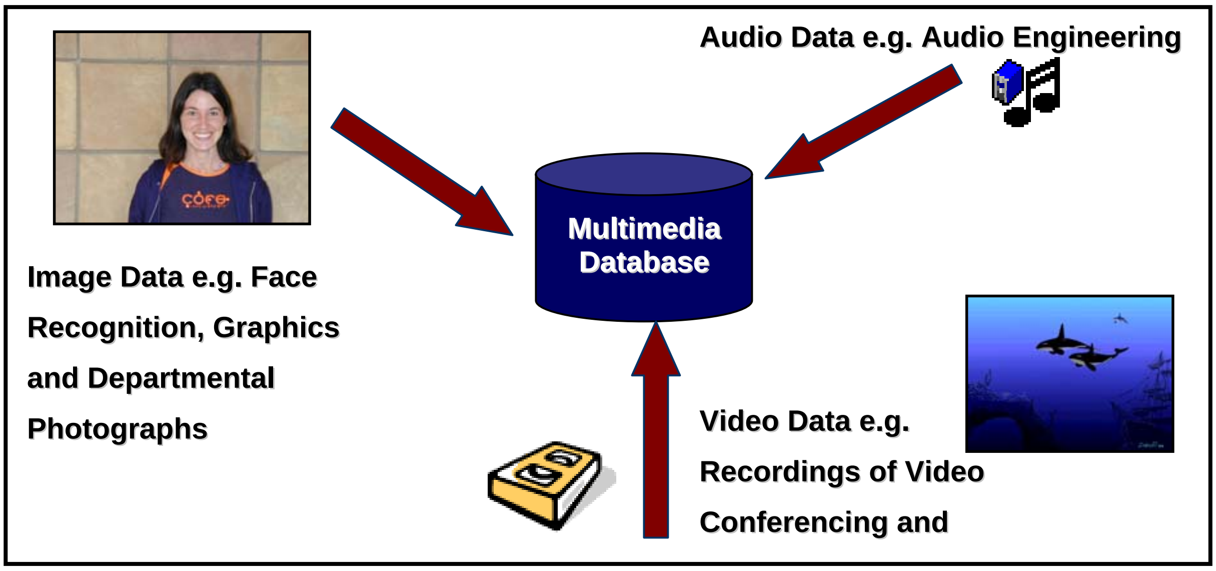

Audio Data e.g. Audio Engineering
Multimedia Database
Image Data e.g. Face Recognition, Graphics and Departmental Photographs
Video Data e.g. Recordings of Video Conferencing and

Figure 5-1 Types of multimedia data produced in the Rhodes Computer Science Department

Multimedia data varies both in size, ranging from images of a few kilobytes to video data of many hundreds of megabytes (in the Computer Science Department at Rhodes University), and format, ranging from images used for virtual environments to the departmental photographs (in the Computer Science Department at Rhodes University). MORD must therefore test a representative range of formats and sizes for image, audio and video data. All the standard functionality (insert, update, select, and delete) is required by the application, in addition to some advanced functionality. Examples of the advanced functionality are signature generation, to enable content-based retrieval, and scaling of images, to generate thumbnails. MORD should either already be able to, or it should be easy to extend it to, evaluate the performance of this functionality.

5.3 Design of MORD

This section details the design of MORD’s schema, test data, workloads, and performance metrics, and describes the manner in which it has been coded.

5.3.1 Schema Design

MORD is a domain-specific benchmark. The schemas of domain-specific benchmarks can either be modelled on a specific application or a more generic database within that domain. The former approach can be used if the applications in that domain are closely matching, and thus a schema designed for one application would be adequate to test most of the applications. For example, the requirements of most engineering applications are similar and the schema of the 007 benchmark is modelled on a specific engineering application. Examples of the latter approach are the SEQUOIA 2000 and the FTR benchmarks, which are modelled after more generic applications. The schema of SEQUOIA 2000 was modelled on an Earth Science (ES) application, yet it is generic enough to be used for most ES applications, and the schema of FTR was modelled on a generic application to store and search for documents. MORD evaluates multimedia databases, and as such the schema needs to model a multimedia application. MORD’s schema is designed to represent a generic multimedia database, with several tables to hold multimedia data of more than one type.

A script achieves the generation of MORD’s schemas, so that it is easy to create or recreate the tables in the database, making the alterations to the database. The script, given in Appendix A, creates a separate table for each of the multimedia datatypes (image, audio and video) with three attributes in each of these. The reason for using disconnected tables is that, generally there is no connection between the different multimedia formats. This is similar to the nature of the data in SEQUOIA 2000, which has four independent tables. In addition, MORD aims to measure the basic functionality of a multimedia database and this does not include the synchronisation of different multimedia types which would require linked tables.

As with most benchmarks, such as BUCKY and 007, there is an identifier (id) for each table which uniquely identifies each entry. In every table, there is also a description of the multimedia data, to categorise the data by size and format, and the actual multimedia data. Figure 5-2 presents the schema used in MORD.

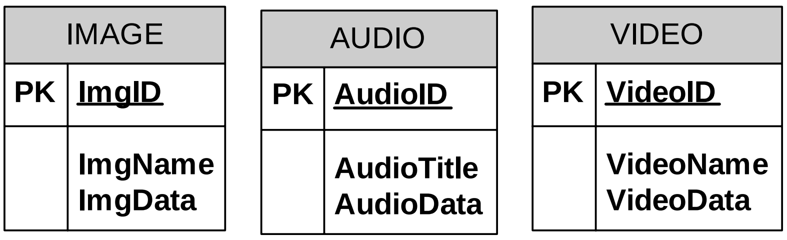


Figure 5-2 Schema used for multimedia tables

Oracle's `NUMBER` datatype is used for the identifier, which is then controlled using `SEQUENCES` as shown in Figure 5-3.

```
CREATE SEQUENCE VIDEO_sequence
INCREMENT BY 1
START WITH 1
NOMAXVALUE
NOCYCLE
CACHE 10;
```

Figure 5-3 Oracle code to generate a sequence

This datatype can be changed to the corresponding primary key datatype when applying MORD to a different DBMS. The description attributes (name or title) uses the standard datatype `char(50)`, found in most DBMS. Finally, the multimedia datatype are Oracle's *inter*Media datatypes: `ORDSYS.Image` for images, `ORDSYS.Audio` for audio, and `ORDSYS.Video` for videos. These datatypes are unique to Oracle and they must be altered to the equivalent multimedia datatypes for alternative DBMS. The script includes code to drop the tables before recreating them in case the tables already exist as the database prohibits the creation of tables with the same name.

Two additional tables are provided and can be included if required. One of the tables is for documents, while the other one is for image data with the addition of a column for thumbnails, as shown in Figure 5-4.

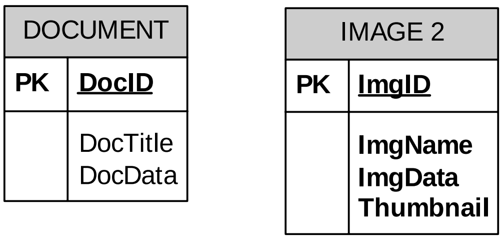


Figure 5-4 Additional tables included

These tables are used in tests to show that MORD's schemas can be altered.

If a benchmark aims to determine the maturity of a new technology, such as BUCKY, it needs to include the schema implemented in the older database technology with which to compare the results of the new database. It may appear that MORD evaluates the performance of a new technology, ORDBMS. This is not the case though as MORD's focus is on the handling of multimedia data and not on evaluating ORDBMS as a new technology. Since MORD is not such a benchmark, it does not need to provide additional schemas, such as relational schemas.

5.3.2 Test Data

A number of factors influence the design of test data: the method of generating the data, the type and format of the data, and the amount of data.

5.3.2.1 Generation of Multimedia Data

The test data must include unique primary keys, descriptions of the data, and a range of multimedia data. Many benchmarks synthetically generate the test data, as it is an easy and reliable method, but this test data is usually only simple datatypes, such as numbers. The nature of complex data makes it much more difficult to generate synthetically. Multimedia data, in particular, does not conform to any clear pattern and it is also complex in structure, making it difficult to generate. Evidence of this comes from existing benchmarks that include some multimedia data, for example, the SEQUOIA 2000 benchmark, FTR benchmark, and TPC-W benchmark, where the multimedia data has been provided rather than synthetically generated. Similarly, MORD's multimedia test data consists of pre-existing data.

Although MORD's multimedia data is provided with the benchmark, the remaining test data is synthetically generated. The first datatype that is synthetically generated is the description. This data is produced using a template that has the corresponding category that the data belongs to substituted where required. The purpose of data is to easily identify the category to which the data belongs. The second datatype, which could possibly be classified as synthetically generated although it is not clear-cut, is the id. This data is generated using Oracle's `INSERT.NEXT_VAL` method, which increments each new entry by 1, in conjunction with its `SEQUENCE` datatype, which is already implemented in the schema. These ensure a unique value and a uniform distribution of the data throughout the table.

5.3.2.2 Format of Multimedia Data

MORD differs from other benchmarks in that its test data includes a wider range of multimedia datatypes and formats. Its multimedia test data includes images, audio, and video data, which does not have to be unique, but must consist of a representative selection of data.

The formats that Oracle supports dictate the selection of multimedia data. (This support can be extended through cartridges, an aspect outside the scope of this thesis.)

Oracle supports most image formats. Of these, *GIF, JPEG, BMP, TIFF,* and *PNG* are included in the test data. CompuServe developed the Graphic Interchange Format (GIF) file format in 1987. The Joint Photographic Experts Group (JPEG) file format is named after the group that created it and was standardised by ISO in 1990 [JPG02, W3C02]. The Bitmap (BMP) file format is a graphics format that was originally designed to be used on the Windows platform [STA98]. The Tag Interchange File Format (TIFF) file format is a tag-based format that was originally developed by Aldus (now Adobe) [STA98]. Lastly, the Portable Network Graphics (PNG) file format is the successor to the GIF format with the significant difference that it is patent-free [PNG02]. The characteristics of these image formats are given in Table 5-1.

Format	**Compression**	**Works Well For**	**Main Difference**
GIF	Lossless Compression	Few Colours	Display Transparency
JPEG	Lossy Compression	Natural Scenes	Great Compression
BMP	No Compression	Windows Platform	Format Reversed
TIFF	Lossless Compression	None in Particular	None
PNG	Lossless Compression	Few Colours	Patent-Free, Portable

Table 5-1 Characteristics of image data

GIF, PNG and TIFF all use lossless compression, while JPEG uses a lossy compression and BMP does not use any compression. GIF and PNG are both highly compressed graphics format using the same type of compression, where the degree of compression is dependant on the amount of repetition in the image. For example, an image with a few distinct colours compresses well [STA98]. One of the advantages that GIF files have over most other image formats is that they can display transparency so that, for example, the background colours can show through an area of an image. TIFF images can be compressed using the lossless LZW (Lempel-Ziv Welch). JPEG uses a lossy image compression and thus the decompressed image is not exactly the same as the original image. This compression is achieved by saving only the relevant colour information essential to the image in an RGB image, with the result that greater compression can be achieved. JPEG is most effective for natural or real-world scenes, such as photographs, and can compress either full-colour or grey-scale images [W3C02]. BMP are not compressed and consequently the images are generally larger than alternative formats [STA98]. The BMP format differs significantly from the other formats in that it stores image data in the reverse order (from bottom to top and pixels in the order blue/green/red).

Oracle only supports four audio formats, MP3, WAV, AU and AIFF, and of these the first three are part of the test data. MPEG-1 Audio Layer III (MP3) was originally developed by the German institute Fraunhofer Gesellschaft (FhG*)*, which still holds its key patents [FRA01]. It was introduced as a part of the official MPEG-1 standard in 1992 and is largely thought of as the most successful audio-standard since WAV. Microsoft and IBM developed the Waveform (WAV) audio format and it was one of the first audio formats, while the NeXT/Sun (AU) audio format is a common compressed file format used for UNIX [ADO02]. Table 5-2 shows the characteristics of each of these formats.

Format	**Compression**	**Advantages**	**Platform**
MP3	Lossy Compression	Small, Can be Streamed	Any Platform
WAV	Compressed or Uncompressed	Excellent Sound Quality	Windows
AU	Compressed	Widely used on Internet	UNIX

Table 5-2 Characteristics of audio data

The MP3 format uses a very powerful type of compression, capable of reducing file sizes down to about 1MB a minute [ADO02]. Although this is a lossy type of compression, the sound quality is still excellent as it only removes information that is mostly beyond the perception of the human ear [ADO02, and FRA01]. This format is also designed to be easy to stream. WAV is the common audio file format used for Windows and since it offers one of the best sound qualities it is a popular format. Unfortunately, this sound quality comes at the expense that WAV files are large in size whether compressed or uncompressed. For example, they can be up to 10MB per minute. AU is not a popular audio format, although recently it has been used more on the Internet.

Oracle only supports three video formats, and all three are used in the test data. They are AVI, MOV and RM Surprisingly enough, Oracle does not support the MPEG video format, which is a very popular, widely used video format. The Audio Video Interleaved (AVI) video file format was defined by Microsoft and conforms to their Windows Resource Interchange File Format (RIFF) specification [MCG02]. This format is called AVI because it interleaves audio and video segments. AVI is one of the most common video formats on the PC, where it has become the *de facto* standard for video data. MOV on the other hand was developed by one of Microsoft's competitors, Apple Macintosh, and is a QuickTime movie. QuickTime movies consist of three layers, namely the Movie file, the Media Abstraction Layer, and the Media Services. RealNetworks originally developed an audio format known as RealAudio, which was later followed by a video format known as RealMedia (RM) video file format [SON00]. This format is a network based format that was originally designed for the Internet. It allows the video to be streamed at low bandwidth, and thus with reduced quality.

5.3.2.3 Amount of data

Many benchmarks provide different sized databases. For example, the EDB benchmark has a standard, large (standard x10), and huge (standard x100) database, and the 007 benchmark has a small, medium, and large database. MORD also has three database sizes: small, medium, and large. Table 5-3 shows the total size and the data ranges of each item for each category of database.

Database Size		**Image**	**Audio**	**Video**
Small	**Range**	1KB – 50KB	1KB-500KB	300KB- 3MB
	Total	2.5MB	15MB	75MB
Medium	**Range**	50KB – 500KB	500KB- 5MB	3MB-30MB
	Total	25MB	150MB	750MB
Large	**Range**	500KB - 5MB	5MB – 50MB	30MB – 300MB
	Total	250MB	1.5GB	7.5GB

Table 5-3 Amount of data stored in each table

There is an increase in size by a factor of 10 between the small and medium, and medium and large database sizes, as it can be seen from Table 5-3. To make the total data size there are approximately twenty entries for each category. The data is distributed so that the first five items add up to the same size as the next five items, and all twenty items total the values given in Table 5-3. MORD differs from other benchmarks in that its total size of test data is much larger than that found in others.

5.3.2.4 Testing

It was tested that Oracle supports the format of all of MORD's test data by using Oracle's `setProperties()` method, which reads and interprets the metadata (properties encoded in the header) of the multimedia data. Since this metadata is made accessible to the database, other methods could then be used to display the metadata and verify that Oracle supported that data format. It was also verified that each category contained the correct number of items by attempting to execute the code to insert the data. If that category contained an incorrect number of items, an error message was printed to identify the data that was causing it. These messages utilise the naming system implemented where each file is named with their category and a consecutive numbers. For example, the first bitmap image in the small database is called *sm1.bmp*.

5.3.3 Workloads

Previous benchmarks have shown that workloads should either represent a specific application, or the important functionality required by a type of application. MORD's workload tests the latter since it targets a generic multimedia database. As done by the Michigan and Wisconsin benchmarks, MORD tests primitive queries including insert, select, update, and delete. However, they have to be implemented using methods adapted for multimedia data. Several additional operations, unique to multimedia data, are also included in the benchmark further distinguishing MORD's workload. The unique implementation of the primitive queries as well as the special functionality makes MORD's workload very different from the workloads of other benchmarks.

5.3.3.1 Insert Queries

The insert query is especially important as it determines whether the database is a "true" multimedia database. With many multimedia databases, the insert operation only creates a reference to the multimedia data, which remains stored on the file system. A multimedia database should import the multimedia data into the database so that it is under the full control of the DBMS. The insert operation is performed in a number of stages. The first stage is executed using a standard SQL `insert` statement, as shown in the example code for the image data in Figure 5-5.

```
INSERT INTO image(imgid, imagedata, imagename)
      VALUES (image_sequence.NEXTVAL,
      ORDSYS.ORDImage.init(),
      'Images of " + imgNm + " type:"+ imgType + "')")
```

Figure 5-5 Code to insert multimedia data into a database

In this statement there are datatypes, such as `ORDSYS.ORDImage`, and methods, such as `NEXTVAL()and INIT()`, that are unique to Oracle. Oracle's `NEXTVAL()` method ensures that the primary key is unique. Oracle's `INIT()` method initialises the multimedia object datatype, but it does not upload the data. To upload the data Oracle offers several alternatives, including the `loadFromFile()`, `loadFromStream()`, `loadFromByteArray()`, and `import()`

methods. Of these, `loadDataFromFile()`was chosen for no particular reason other than its simplicity. `setProperties()`is used to extract the metadata from the multimedia data and store it in the database. If MORD is applied to another multimedia database, the methods to initialise the object and upload the multimedia data as well as the multimedia datatypes have to be modified.

5.3.3.2 Select Queries

The select query is required to retrieve the data, and again this has to be performed using special code for multimedia data as opposed to the standard `select` statement used for ordinary data. Multimedia data not only needs to be retrieved, but it also needs to be presented correctly, which takes additional processing. Oracle implements such a task using `getContent()`, which would have to be altered for a different multimedia database.

5.3.3.3 Update Queries

The update query could either alter or completely replace the data in a database. MORD only evaluates the latter, where the time taken to replace the multimedia data with the entry in the row following it is measured. The first step is to select the data that is to replace the old data, which is achieved by using the code given in Figure 5-6.

```
OracleResultSet rs = (OracleResultSet)stmt1.executeQuery(
      "select imagedata from image
      where imgid=" + (imgid + 1) + "
      for update");
```

Figure 5-6 Code to select data with which to replace the old data

Here it can be seen that the data from the row with `imgid + 1` is being selected. The next step is to replace the multimedia data of the row corresponding to `imgid` with this data. This is accomplished using the code shown in Figure 5-7.

```
OraclePreparedStatement stmt2 =(OraclePreparedStatement)
con.prepareCall("update image
                  set imagedata= ?
                  where imgid =" + imgid);
stmt2.setCustomDatum(1,imgobj);
stmt2.executeUpdate();
```

Figure 5-7 Code to replace the multimedia data

The metadata is extracted from the multimedia data so that each row contains the new properties.

5.3.3.4 Delete Queries

Data that is no longer relevant should be deleted. Such a task is more complex for multimedia data as it involves not only deleting the data from the table, but also freeing the space in the tablespace that data occupied. Freeing the space is important with multimedia data as it often occupies a large amount space. It can be achieved using Oracle's `deleteContent()` method, followed by the standard SQL `delete` statement, as given in Figure 5-8.

```
while (rs2.next())
{
      OrdAudio audobj =
      (OrdAudio)rs2.getCustomDatum(1, OrdAudio.getFactory());
      audobj.deleteContent();
      OraclePreparedStatement stmt2=
            (OraclePreparedStatement)con.prepareCall(
            "update audio set audiodata= ? where Audioid =" + audid);
      stmt2.setCustomDatum(1,audobj);
      …
}//while

stmt =(OraclePreparedStatement)con.prepareStatement("
      delete from image where imgid="+ imgid);
```

Figure 5-8 Code to delete data from the table

`deleteContent()`frees the storage space used by the multimedia data, but this method would have to be replaced by the equivalent method for a different multimedia database.

5.3.3.5 Testing

Tests were performed to check that the code for inserting the multimedia data was written correctly. Initially tests were performed to ensure that the data was actually inserted into the database and that the properties had been extracted. Firstly, the increase in the used space for the tablespace was analysed to validate that it is proportional to the size of data being inserted. Secondly, the original data was relocated and attempts were subsequently made to retrieve the multimedia data from the database. This showed that the data was physically stored within the database and that Oracle was not simply using pointers.

To test that the selection process was executed properly, the Java Media Framework (JMF) and the Java Media Framework Registry (JMFReg) were installed and the necessary DataSources were registered. Additional code was run to play the audio and video data, ensuring that it was being retrieved properly.

The properties could be used to verify that the data was updated by displaying them before and after the update. It could be checked that the new properties matched the values previously assigned to the preceding row. Another method used to verify that the update was performed correctly was viewing the data.

Checks were also performed to ensure that the data was deleted from the table and that the space had been freed. It was easily verified that the entries were deleted from the table by using a "`select id from table`" SQL statement which showed if there were any entries remaining in the tables. It was more difficult to verify that the data had been deleted from the tablespace. DBMS handle the freeing of space no longer in use in several different ways. Oracle handles this by marking the data as deleted and only reusing that block when the tablespace is full and needs the space. (This is the default setting, but changing the value of certain tablespace parameters can alter it.) To verify that the data was deleted correctly, the first step was to identify the location of the data in the tablespace prior to executing the `deleteContent()` method. This was achieved by viewing the tablespace map and finding the block identifiers detailing where the data was stored. The next step was to execute `deleteContent()`. The final step was to query the `dba_free_space` table to find out which blocks in the tablespace were marked as free. The space was freed correctly if the block identifiers matched those of the deleted data.

5.3.4 Performance Measurement

MORD's performance measurements require that decisions are made about performance metrics, timing precision, and the method of timing the operations.

5.3.4.1 Performance Metric and Precision

A performance metric can be thought of as the value used to measure how well a particular operation is executed by the system. Two important measures of performance are response time, which is the average time taken for each query to execute, and throughput, which is the amount of work accomplished in a given time and so is a measure of the overall performance of the system. Response time and throughput are closely related: when the response time for an average query increases, the overall throughput often decreases and vice versa. In MORD, both of these are measured.

It was decided that the response time experiments should be measured in milliseconds to obtain accurate results, but these results were then converted to seconds for analysis. Since the manipulation of multimedia data usually takes longer than other datatypes, due to its large size, it is not necessary for its measuresments to be as fine. As a result, this precision is sufficient. The throughput experiments are measured in queries/sec, but if necessary their results can be converted to Mb/sec as the amount of data/query is known.

5.3.4.2 Timing Mechanism

The Java method provided by the `System` library called `currentTimeMillis()` is able to accurately time the operations by retrieving the time from the system clock and converting it to milliseconds. This Java method is used for MORD, where the code for the response time is shown in Figure 5-9.

```
ts3 = System.currentTimeMillis();
                  for (int i=1; i <=num; i++)
                  { …          }//for
ts4 = System.currentTimeMillis();
```

Figure 5-9 Code to control the measurement of response time

The variable `num` is determined by the user to control the number of entries measured. For example, the user could measure the response time for 10 insert operations or 20 insert operations, depending on the value assigned to `num`. The time limit for the throughput experiments is controlled by the code and uses the `currentTimeMillis()`method, as used for the response time experiments. The code for the throughput experiments is shown Figure 5-10.

```
int count=1;
ts3 = System.currentTimeMillis();
ts4 = System.currentTimeMillis();
while ((ts4-ts3)<timing)
      {     …
            count++;
            ts4=System.currentTimeMillis();
      }//while
```

Figure 5-10 Code to control the measurement of throughput

The user specifies the value for timing before executing the code. The code then keeps looping until the maximum specified time is reached. The number of queries that are performed in that time are recorded using a `count` variable. This value is then decreased by one as it would have only exited when the time was larger than the maximum specified time.

Since the data is always entered in the same order, the amount of data processed in that time is then calculated by adding the sizes of the data up until the `count-1` value.

5.3.4.3 Testing

Several tests were run to verify that the performance measurements were correct. Firstly, tests were performed to confirm that milliseconds are a suitable unit. Running trial experiments and collecting the results to check that they had adequate variation among them achieved this. Furthermore, tests were performed to check that the base case, where no data was manipulated, produced a time of 0ms. Finally, the correctness of the throughput code was shown by printing the time out at various points during the code to check that it had not exceeded the maximum time at any point before exiting. These results showed that the duration of the experiment at various stages during the throughput experiment never exceeded the maximum time.

5.3.5 Programming Language

MORD is written in Java. Java was chosen primarily for its portability, as Java is designed to run on multiple platforms with minimal difficulty, and its compatibility with Oracle, since Oracle is designed to be integrated with Java and so the multimedia methods are written in Java. Additional features that make Java attractive to use are standard programming language features, such as its simple timing methods, its ability to print the results to a file so that there are permanent copies of the results, and its database functionality making it easy to connect to the database. Another option was to use PL/SQL, which is Oracle’s procedural language combined with SQL. Although this programming language provides all the functionality needed and works well with multimedia data, it was not used as it would limit the benchmark to Oracle only.

5.4 Application of MORD

This section gives an overview of the issues that should be considered before applying MORD. It also describes the typical results that can be expected from running MORD and explains the way in which these results can be analysed. Further instructions for the application of MORD are given in Appendix B. Before applying MORD to a system, the complete environment, including the database and its surrounding system, needs to be setup. It is essential that the environment in which MORD is run is kept constant. Consideration must be taken of how to control the external influences as well as ensuring that the internal environment, the database and its immediate environment, remains identical throughout the experiments. One of the most significant factors when considering the external influences is the effect of being connected to a network. A network connection has a negative influence on the environment, particularly its high level of variability, which makes results difficult to interpret. The network should be excluded from the environment by running both the client and database server on a single machine disconnected from the network. Although this limits the results to an isolated database, it ensures that the network and other potential external influences have no effect on the results.

The immediate environment, which includes the physical environment and the manner in which the experiments are run, is more difficult to keep constant. An important influence in controlling the physical environment is to ensure that there are no other significant processes running on the machine using the CPU and memory while the experiments are executed. Unnecessary services can be stopped by using the administrator tools of the operating system, whereas the processes can be closed by using the task manager.

There are several steps that must be taken to control the application of MORD to ensure the accuracy of the results. The first one is to run the experiments several times to fill the cache so that the experiments are run on what is known as a *hot* database, a database with full caches. An indication that the system is stable is where consecutive results from running MORD produce closely matching values. This problem not only occurs with the data cache, but also with the Java cache, as it is shown by the fact that the first query takes longer as a result of the Java code being cached. Benchmark variables that can be altered must be kept constant, unless being tested themselves, such as the number of users should be 1 and the number of data items should be 20 for image and 5 for audio and video. To keep the database environment constant a default configuration should be used for all the experiments with only the parameter under observation altered for each new database configuration tested. Once the environment has been correctly setup, MORD is applied to the various test systems to produce results.

MORD consists of 4 query types, 3 multimedia datatypes, 2 performance metrics, and 3 database sizes, thus producing a possibility of 72 results. Usually only a selection of these options is used or further calculation is used to reduce this number. Each experiment in MORD should be run multiple times, after which the arithmetic mean or average of the results from these experiments is calculated to produce a single value. Ideally, the more times the benchmark is run the more accurate the results. Since the results are printed to a Microsoft Word document file, a permanent record of each implementation can be kept by appropriately renaming the file after each experiment. Alternatively, additional code can be added to print information in the result file that identifies which test the results belong to.

Finally, after applying MORD, it must also be decided how the results should be presented. This includes whether the final results of MORD should be as many values or a single result. It was decided that the benchmark should return numerous values as one single value is too rough a measure and is mainly used to rank databases and not to tune them.

5.5 Summary

MORD is a domain-specific benchmark that is designed to evaluate the performance of multimedia databases. Since a good domain-specific benchmark should be portable, simple, scalable, and relevant, MORD is designed to have these characteristics. It is also designed to be suitable to help with the configuration of the multimedia application that is required by the Computer Science Department at Rhodes University.

The design of MORD includes the specification of the test data, schema, workload, performance measurements, and programming language. The test data includes images, audio, and videos, each of which belongs to a small, medium, or large dataset. The schema includes three tables, one for each of the three multimedia datatypes tested, as well as two additional tables to test documents and advanced functionality for images. The workload includes code to test the insert, select, update, and delete operations, in addition to more advanced code to generate thumbnail images.

MORD measures response time in milliseconds, and throughput in queries per second. Timing is determined using Java code and the system clock. MORD is implemented in Java for its portability and its close relationship to Oracle.

When MORD is applied to a system, the environment must be kept constant. Results are made more accurate by repeated application of MORD.

Chapter 6 : Validation of MORD

Chapter 5 described the design of a new multimedia database benchmark, MORD. This chapter explains the design of tests to verify if MORD achieves its objectives. The tests are grouped into categories to test if MORD is relevant, adaptable, and simple to use and extend. Where appropriate, informal hypotheses based on general database theory are also tested. Such tests help increase the understanding of multimedia database performance as they indicate that what holds for standard data also holds for multimedia data.

6.1 Overview of Validation

Validating a benchmark is done by demonstrating that the benchmark achieves its objectives. The objectives of MORD are that it is relevant, adaptable, and simple. Relevancy refers to the fact that the benchmark tests common and important situations that would be found in real applications. Adaptability refers to the fact that the benchmark is flexible enough to adapt to test the specific requirements of an application. MORD is adaptable, if it is both portable and scalable. Simplicity refers to the fact that the benchmark is easy to adjust to suit different environments as well as being straightforward to extend, thus allowing for the addition of extra features when necessary.

A suitable approach to validation is to design tests based on general database theory that have a known outcome. In the case of MORD, general database theory is used to form informal hypotheses that have been proven for ordinary data in Oracle and which are likely also to hold for multimedia data. The results of validating MORD using this strategy, not only indicate that it is designed correctly, but also provide grounding knowledge on basic functionality in multimedia database performance.

6.2 Methodology

Altering the database configuration and/or the benchmark configuration and executing MORD in a controlled environment perform the validation tests. The database's configuration is altered through Oracle's user interface, while MORD's configuration is generally altered manually by changing the code except for when the test data is altered. When the benchmark code is altered manually the instructions in Appendix B must be followed. This task is made easy by comments in the benchmark code that clearly mark the places in which to make such alterations.

Each test is run multiple times until five results that are relatively close are recorded, and any other results are disregarded. This process eliminates random results that are most likely caused by background database operations. The results are collected and statistically analysed, generally by calculating the arithmetic mean although other methods such as standard deviation are also used. The analysis of the results is used to give an overview of MORD's capability and a broad idea of the performance of Oracle's multimedia database. A more in-depth analysis is, however, performed for selected aspects of the benchmark, such as the inserting of images, to provide a greater understanding of the performance in these areas.

6.3 Environment Setup for Validation Tests

Common to the presentation of benchmark results is the detailed description of the hardware configuration. This is crucial if absolute values are measured and to some extent less important in a comparative study. Even though the work done in this thesis aims to test that MORD works correctly rather than to obtain performance results, the hardware configuration is still specified to give some perspective on the results. The environment in which the benchmark is validated only needs to be configured after designing the experiments, but it is discussed here since some of the information in this section, particularly regarding Oracle's configuration, relates to decisions made during the designing of the tests.

6.3.1 Hardware Configuration

The main hardware details that need to be presented are the amount of memory, the processing power, and the physical storage. It is not necessary to have a full scale multimedia database to validate MORD, and as a result the hardware used in the test environment is smaller than that required by an average multimedia application. A summary of the hardware configuration is given in Table 6-1.

Hardware Component	**Type**	**Configuration**
Memory	Standard	1GB
Processor	Dual processor - PIII	800MHz
Hard Drive	IDE	100GB

Table 6-1 Summary of hardware configuration for test environment

It can be seen that 1GB of memory is used for the test environment, which is double Oracle's minimum recommended value of 512Kb. This amount of memory is large as Oracle requires a sizeable amount of memory for all applications, and particularly when implementing a multimedia database.

The rapid advances in the processing power of computers have led to it being less likely the bottleneck in database operations. A reasonably fast processor is thus sufficient, except for the occasional application that has highly CPU-intensive operations. A dual PIII processor at a speed of 800MHz was used to test MORD, which appears to be adequate as the utilisation of the processor was for the most part below the 50% threshold during the tests.

In contrast to the processor, the configuration of the physical storage, both in terms of quantity and type of storage, is significant since disk I/O is often the bottleneck of a database. This is particularly important when dealing with multimedia data, on account of its typical large volume. The environment for testing MORD only has a 100GB IDE hard drive, although the implementation of multimedia applications should have much larger hard drives and should be configured using a RAID system.

6.3.2 Oracle Configuration

Oracle's configuration is controlled by the settings of numerous parameters. Oracle assigns default values to them based on the most suitable value for the average application. Since multimedia applications differ considerably from standard applications, these default values are often unsuitable and need to be changed. Some of Oracle's parameters can be altered dynamically whenever necessary, while others only allow their value to be changed when the database is created. Table 6-2 summarises the significant parameters for the work done in this thesis.

Parameter	Function	Default	Possible Values	Dynamic
SGA_MAX_SIZE	Maximum size of SGA for the lifetime of the instance	Dependent on Pools	0MB - OS Specific	No
DB_BLOCK_SIZE	Data block size	2K	2K-32K	No
DB_CACHE_SIZE	Size of buffer cache	48MB	1 granule upwards	Yes
SHARED_POOL_SIZE	Size of the shared pool	64MB	300Kb - Os Specific	Yes
JAVA_POOL_SIZE	Size of Java pool	20KB	20KB – 2GB	Yes
LARGE_POOL_SIZE	Size of Large pool for shared server session memory	0MB	0MB to 2GB	Yes
OPEN_CURSORS	Max number of open cursors session can handle	50	1 to 4294967295	Yes
Logging	Determines if logging is performed	LOGGING	LOGGING/ NOLOGGING	Yes
Caching	Determines if caching is performed	NOCACHE	CACHE/NOCACHE	Yes
LOB Cache	Determines if caching of multimedia data performed	OFF	ON/OFF	Yes

Table 6-2 Oracle parameters that are altered

DB_BLOCK_SIZE and SGA_MAX_SIZE are both static parameters that must be set when the database is initially created. The default value of DB_BLOCK_SIZE is too small and therefore it is increased to 8KB for these experiments, which improves the performance when dealing with multimedia data.

The value of SGA_MAX_SIZE must be as large as the total size of the pools contained in the SGA (the sum of the buffer cache, shared pool, java pool, and large pool). It is set to 400MB for these experiments, which allows for variation in the sizes of the pools, such as the DB_CACHE_SIZE and the JAVA_POOL_SIZE. The sizes of these pools can easily be altered as the parameters controlling them are dynamic. The DB_CACHE_SIZE is increased to 200MB since the default value is too small to handle the large volume of multimedia data. The JAVA_POOL_SIZE is increased to 40MB as the code is written in Java and may exceed the default storage reserved. The values of the SHARED_POOL_SIZE and LARGE_POOL_SIZE are not changed as their values are of less importance given that the experiments are run in dedicated mode.

Logging and caching are implemented in most databases to enable recovery and speedup certain operations in the database. The default setting, which is to have them turned on, is used for all of the experiments, except the experiment to explicitly test the effect these values have on the performance.

6.4 Relevancy Tests

6.4.1 Design of Relevancy Tests

MORD is designed to be as relevant as possible, while only focussing throughout on basic, but essential functionality, as well as including test data consisting of common data formats. A principal requirement of a relevant benchmark is to be accurate, and thus it is necessary to test that MORD is accurate. Tests at each stage during the design of MORD indicate that it is designed correctly, but tests are still needed to show the accuracy of its overall operation. Unfortunately, proving the accuracy of MORD is made difficult by the fact that an alternative benchmark that measures the performance of multimedia databases is not available with which the results of MORD could be compared. Instead, evidence of it being accurate come from demonstrating that the manner in which the performance is timed behaves as expected in addition to showing that the results obtained from MORD are repeatable. For example, it must be tested that the time recorded for multiple queries is equivalent to the total time that would be recorded if each individual query was timed. Additional evidence that MORD produces correct results comes from the assessment of hypotheses in the tests that follow. In general, results that are as expected, or whose variation is explainable, suggest that MORD is accurate. These methods do not conclusively prove that MORD is accurate, but together they strongly support the likelihood that it is.

Additionally, MORD is relevant in that it allows the user to select the operation, datatype, and repetition in the dataset that are tested. This feature is particularly important in a benchmark, for example, as a user might have an application where it is only relevant to test the insertion of video data.

6.4.2 Test 1: MORD's Measurements are Repeatable

To illustrate that the measurements recorded by MORD are repeatable, individual results are analysed instead of averaging the results as with the other tests. These results are compared to verify that they are either similar or there is a justifiable reason for their difference. The standard deviation is used to calculate the amount of variation there is between the different results. The formula for standard deviation is:

$$\surd\ (n\sum x^2 - (\sum x)^2/n\ (n\text{-}1))$$

The lower the resulting standard deviation, the more repeatable are MORD's results.

6.4.3 Test 2: MORD's Timing Method is Designed Correctly

Test 2 aims to show that the code used to measure the response time is designed correctly in order to verify that MORD's response time is measured accurately. It must be tested that the time for the whole loop is equal to the sum of the time for performing each individual operation. This is achieved by including additional code to time the individual queries and comparing the times they produce with the total time.

6.4.4 Test 3: Specified Queries Can Be Tested by MORD

The queries that a benchmark is required to evaluate in various contexts are different, and thus MORD should enable the user to select the operations to be tested. Such a feature is necessary, for example, in the testing of bulk loading, as bulk loading influences only the insert query.

Bulk loading is essential when a large amount of existing data is inserted into a new application, either from a file system or an older application. Bulk loading presents a unique situation in that it is once off and can be restarted if problems occur. Two database operations that can be configured differently for bulk loading are data caching and logging. Data caching involves storing data in memory to speed up its subsequent retrieval, while logging involves the tracking of operations in order to recover a database. Oracle allows both of these database operations to be turned on or off. They slow down queries and significantly decrease the database's performance and so should be turned off if not imperative. For bulk loading, they are not necessary.

While demonstrating that MORD can focus on a specified query, Test 3 reflects a simple hypothesis about bulk loading (hypothesis 1):

Turning Off Caching and Logging Improves Performance When Bulk Loading

The results of testing this hypothesis can also be used to show the effect on the performance of caching and logging.

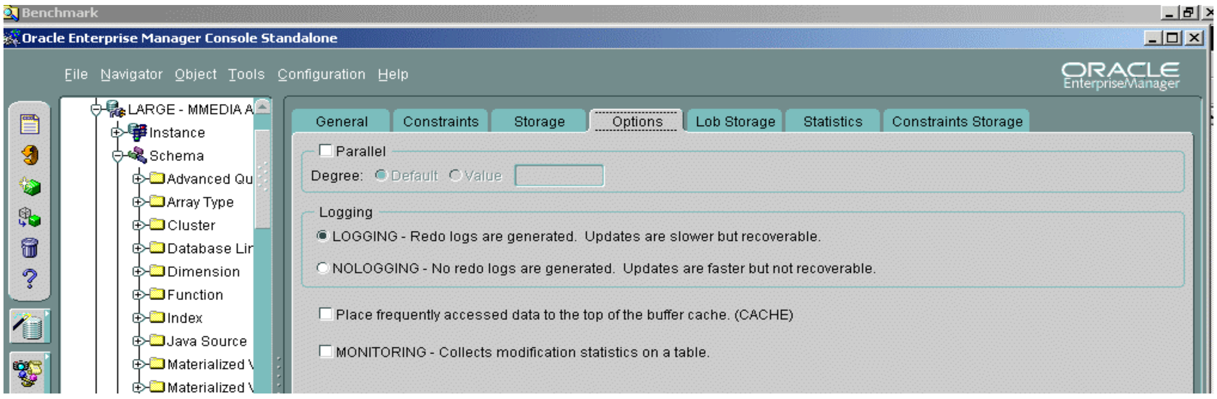
Test 3 investigates the response time for the insert query with all three multimedia datatypes used in this thesis, that is, images, audio, and video. The relevant parameters for testing hypothesis 1 include tablespace parameters `CACHE` and `LOGGING`. The values of the tablespace parameters must either be set to `NOCACHE/NOLOGGING`, or alternatively `CACHE/LOGGING.`

(The combination `CACHE/NOLOGGING` is not permitted by Oracle.) The database configurations used for Test 2 are shown in Table 6-3.

	Datatype	**Operation**	**Performance Metric**	**Parameter Settings**
Database Configuration 1	Image, Audio, Video	Insert	Response Time	`NOCACHE, NOLOGGING`
Database Configuration 2	Image, Audio, Video	Insert	Response Time	`CACHE, LOGGING`

Table 6-3 Database configurations for Test 2

The values of the parameters can be altered using Oracle's Enterprise Manager. A screenshot displaying Oracle's interface to alter these parameters is shown in Figure 6-1.



☐ Parallel
Degree: [● Default] ○ Value
Logging
[● LOGGING - Redo logs are generated. Updates are slower but recoverable.]
○ NOLOGGING - No redo logs are generated. Updates are faster but not recoverable.
☐ Place frequently accessed data to the top of the buffer cache. (CACHE)
☐ MONITORING - Collects modification statistics on a table.

Figure 6-1 Screenshot to show how the caching and logging values are controlled

The expected results from Test 3 include a demonstration that MORD enables the user to select the required queries, as well as results that reveal the extent to which the performance can be improved by turning the logging and caching off.

6.4.5 Test 4: MORD Can Focus on Certain Datatypes

The datatypes that a benchmark must evaluate differ according to the application, and thus MORD should allow the user to select the datatypes that are tested. For example, an application that is designed to store identification photographs of people and contains no other multimedia data would have no need to evaluate the performance of retrieving video data. To verify that MORD can focus on a single datatype, a test is designed in conjunction with the test that follows, Test 5, to evaluate the performance of images only.

6.4.6 Test 5: MORD's Dataset Composition Can Be Varied

Since the dataset that is affected by the queries of an application can vary, it must be possible for the user to alter the composition of MORD's dataset. For example, a situation where the composition of the dataset is important is the investigation of the effect of the buffer cache. The chance of data from the dataset being stored in the buffer cache is determined by the frequency with which the data is accessed. A factor that improves the likelihood of the data being stored in the buffer cache is the size of the buffer cache itself. The larger it is, the greater the amount of data it can store, which increases the probability that the required data is stored in memory. If the data is already in the buffer cache it reduces the physical I/O overhead and improves performance, given that memory access is faster than physical disk access.

However, there is a limit to the performance improvement that can be gained from increasing the size of the buffer cache. Essentially, an increase in the cache size over and above the amount of frequently accessed data would not continue to produce a performance improvement. The general rule is that the buffer cache should, at a minimum, be large enough to store the largest data item found in the database, which is often considerable for multimedia databases. These ideas can be summarise simply by the following formula:

↑BCS = ↑Prf (Greater ↑Prf as DSR approaches 100 and BCS approaches size of DS)

Where: BCS = size of the buffer cache,

Prf = performance of the system,

DSR = repetition in the dataset,

↑ = increase.

This formula implies that the performance increase is greater as more of the data in the dataset is repeated, especially if the size of the buffer cache is large enough to contain the entire dataset. Performance spikes occur when the data is moved out of the cache and replaced by new data, slowing down the performance.

Test 5 reflects a hypothesis based on standard buffer cache theory to demonstrate that the composition of MORD’s dataset has been altered. Hypothesis 2 is as follows:

The Buffer Cache Is Most Effective If

a) The Same Data Is Frequently Accessed and

b) It is Large Enough to Hold This Data

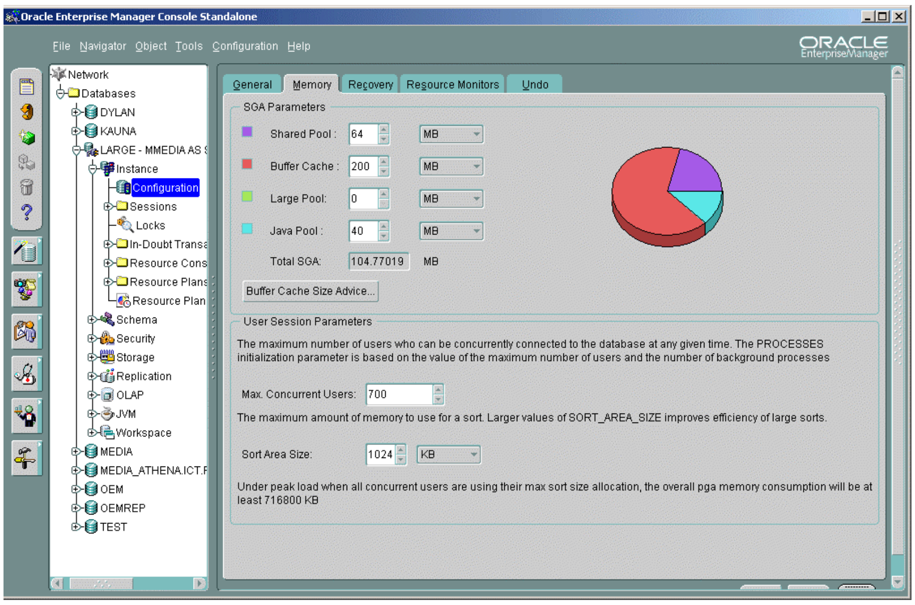
In Test 5, the response time is measured for the select and update operations, since they are the most likely to be affected by the size of the buffer cache. Three different datasets are tested, where the quantity of data repeated in them differs. Dataset 1 consists of only unique items for each query and thus has no repetition. Dataset 2 accesses unique data for a quarter of the test, then for the other three quarters it goes back to the first data item and accesses the same data previously manipulated. Dataset 3 accesses a single data item repetitively.

Since Test 5 is implemented in conjunction with Test 4, it is run for images only. The most significant database parameter for this test is the `BUFFER_CACHE_SIZE`, which controls the size of the main buffer cache. The three values that are tested for this parameter are selected based on the sizes of the image dataset, and they are 16MB, 100MB, and 320MB. The datasets and configurations tested are shown in Table 6-4.

Dataset	**BUFFER CACHE (MB)**	**Performance Metric**	**Operation**	**Datatype**
1	16	Response Time	Insert, Select, Update, Delete	Image
2	16	Response Time	Insert, Select, Update, Delete	Image
3	16	Response Time	Insert, Select, Update, Delete	Image
1	100	Response Time	Insert, Select, Update, Delete	Image
2	100	Response Time	Insert, Select, Update, Delete	Image
3	100	Response Time	Insert, Select, Update, Delete	Image
1	320	Response Time	Insert, Select, Update, Delete	Image
2	320	Response Time	Insert, Select, Update, Delete	Image
3	320	Response Time	Insert, Select, Update, Delete	Image

Table 6-4 Database and benchmark configurations for Test 5

The Oracle Enterprise Manager Console is used to change the value of the buffer cache. A screenshot is shown in Figure 6-2.



SGA Parameters

Shared Pool : 64 MB
Buffer Cache : 200 MB
Large Pool: 0 MB
Java Pool : 40 MB
Total SGA: 104.77019 MB

User Session Parameters

The maximum number of users who can be concurrently connected to the database at any given time. The PROCESSES initialization parameter is based on the value of the maximum number of users and the number of background processes

Max. Concurrent Users: 700

The maximum amount of memory to use for a sort. Larger values of SORT_AREA_SIZE improves efficiency of large sorts.

Sort Area Size: 1024 KB

Under peak load when all concurrent users are using their max sort size allocation, the overall pga memory consumption will be at least 716800 KB

Figure 6-2 Screenshot to show where the size of the buffer cache is altered

The expected results for Test 5 should demonstrate the ability to alter the composition of MORD's dataset. They should also indicate the effect of the buffer cache on the database's performance, as well as the most optimal size to achieve the best performance.

6.5 Adaptability Tests

6.5.1 Design of Adaptability Tests

The adaptability tests show that MORD is portable, scalable, and that the precision of the measurements can be altered.

Portability is not achieved in MORD as explained in Section 1. In an attempt to make MORD as portable as possible, the code is written in Java, a portable programming language. The code further supports adaptability by its modular design, which makes it easy to replace portions of it, and the comments that indicate clearly where Oracle-specific code must be replaced for a different DBMS. MORD is also designed so that it allows the user to select different databases to connect to and evaluate.

To prove conclusively that a benchmark is portable, it would have to be shown that it could evaluate a different DBMS, such as Informix or DB2. The steps necessary to connect to another database include: adjusting the schema, changing the connection string, and altering the Oracle specific code in the methods. Such tests could not be performed because of the unavailability of alternative multimedia databases. However, the steps taken to alter the Oracle database configuration and connect to a different Oracle database are similar to those required when another DBMS is being evaluated. These are therefore tested instead, and although they do not produce compelling evidence of portability, they do demonstrate that MORD has a number of the necessary features for it to be a portable benchmark.

A different Oracle database can either be a database that has been modified, for example, by adding a new column to a table, or a completely new database that has been created from the start and uses different storage space. The ability to alter the design of the benchmark's database has many uses. For example, this capability is required to compare the performance of alternative methods of handling multimedia data, such as BLOBs and Oracle's *Inter*Media datatype. Another example is the evaluation of extra functionality, for instance creating thumbnails or signatures, where columns must be added to tables to store this data. To modify a database, adjustments must be made to the schema while evaluating a completely different database involves the alteration of the connection string that determines the database evaluated by MORD.

A scalable benchmark must have the ability to measure the performance of databases with different amounts of data. Varying the amount of data can be interpreted in two different ways. The size of data can be changed or the number of items in each query can be altered, where in both cases the overall volume changes. Having a benchmark that is scalable is valuable in predicting how the performance of the database will react to an increase in its size, and can be used to indicate the maximum size of a database to attain a certain level of performance. The scalability test demonstrates that the amount of data can be varied, both in number of items and amount of data.

Finally, MORD is adaptable in that the timing method can be altered in terms of both granularity and type. Granularity refers to whether individual tasks or whole operations are measured, whereas the type refers to whether response time or throughput is measured. The granularity test shows that the measurements taken by MORD can be altered according to the requirements of the application being tested.

6.5.2 Test 6: MORD Can Connect to Different Databases

The ability to connect to different databases is a common requirement of a benchmark. For example, if a static database parameter is being examined then a new database is needed for each value tested as the parameter cannot be modified after the database is initially created. Such a static parameter is the `BLOCK_SIZE`, which determines the smallest unit of physical storage, and as a result controls the amount of disk I/O required for each data item (see section 4.4.2 for more information on data blocks). If handling larger data, assigning a bigger value to the `BLOCK_SIZE` should improve the performance by increasing the size of the chunks that the data is divided into.

Test 6 reflects a hypothesis based on theory about the size of the data block to demonstrate that MORD can connect to a different Oracle database. Hypothesis 3 is as follows:

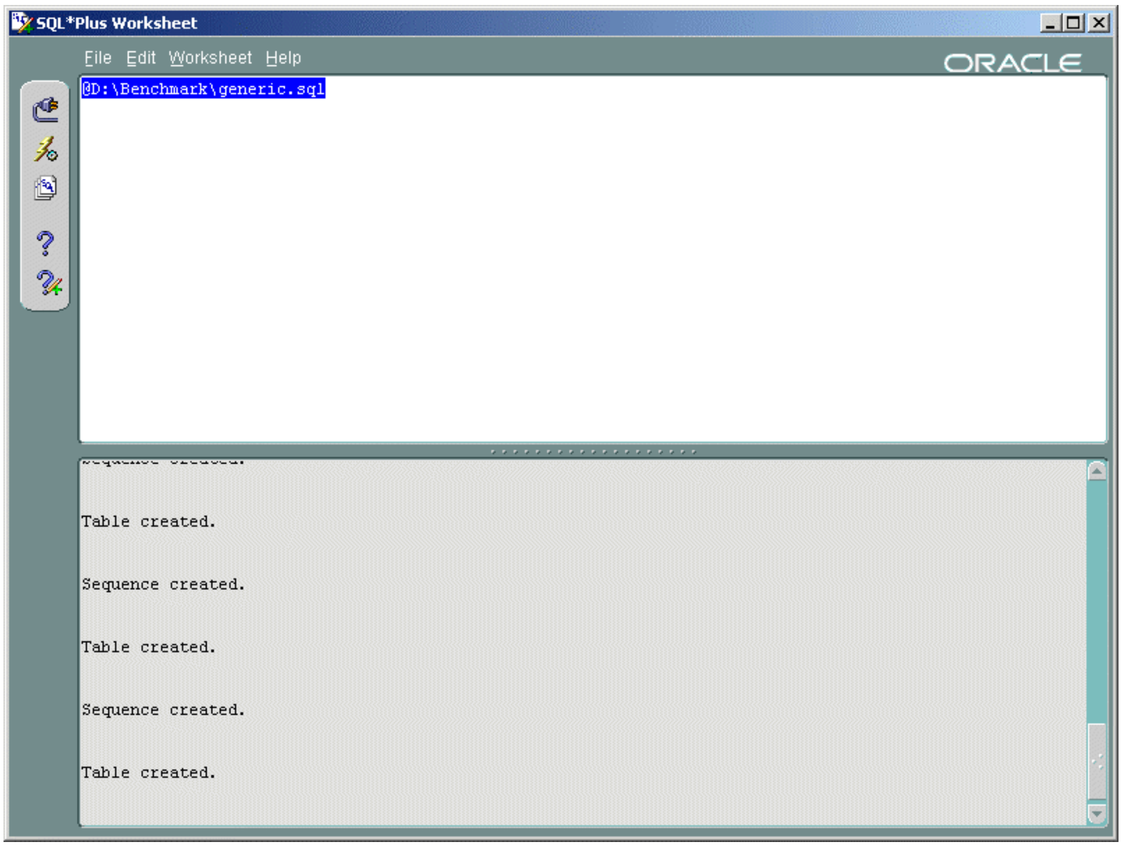
> *An Increase in the Block Size Improves the Performance When Dealing With Large Data*

The response time is measured for all three datatypes, that is, images, audio, and videos, as well as the full set of operations, that is, insert, select, update, and delete. It is sufficient to test only two values for the `BLOCK_SIZE` as the primary aim of Test 6 is to demonstrate that MORD can connect to different databases and not to prove hypothesis 3 fully. These values are 4K and 8K, rather than the default value of 2K which is too small for multimedia data. The database configurations are given in Table 6-5.

	Datatype	**Operation**	**Performance Metric**	**Block Size**
Database configuration 1	Image, Audio, Video	Insert, Select, Update, Delete	Response Time	4K
Database configuration 2	Image, Audio, Video	Insert, Select, Update, Delete	Response Time	8K

Table 6-5 Database configurations for Test 6

The test databases must be created prior to running Test 6. Unfortunately this process is not automatically performed by PL/SQL scripts or Java code provided by MORD, as it is the case with the other tasks. The easiest way to create the databases is by using an Oracle wizard. Once the databases have been created, their schemas must be generated by executing the schema script `generic.sql`. It can be run using the SQL*Plus Worksheet as shown in Figure 6-3. (A complete explanation of this procedure is given in Appendix B).



```
@D:\Benchmark\generic.sql
```

```
Table created.

Sequence created.

Table created.

Sequence created.

Table created.
```

*Figure 6-3 Screenshot of SQL*Plus worksheet to generate database schema*

The database that MORD tests is determined by the connection string consisting of the database name, username, and password. The connection string's value can be specified in MORD's code.

The results are expected to show that MORD can connect to a different database as well as to reveal the extent to which the `BLOCK_SIZE` can improve the performance of the database.

6.5.3 Test 7: MORD's Database Design Can Be Modified

Test 7 demonstrates that the database design can be modified by altering the `generic.sql` script. This test is not run independently, but prepares the database to be used for the simplicity test where MORD must be extended to evaluate the performance of the database when handling documents. The preparation is achieved by modifying the schema to include an additional `SEQUENCE` and a table to support documents. The code to create the `SEQUENCE` is shown in Figure 6-4.

```
CREATE SEQUENCE DOCUMENT_sequence
      INCREMENT BY 1
      START WITH 1
      NOMAXVALUE
      NOCYCLE
      CACHE 10;
```

Figure 6-4 Code to create the document SEQUENCE

The code to create a table for the document data should include a primary key, a column to store the document data, and a column to hold a description of the data. The SQL code to generate such a table is shown in Figure 6-5.

```
CREATE TABLE DOCUMENT(DocID   NUMBER PRIMARY KEY,
                  DocData         ORDSYS.ORDDoc,
                  DocDescription          VARCHAR2(200));
```

Figure 6-5 Code to create a table to store the document data

Once these segments of code have been included in the schema script, the script can be executed and the creation of a new table can be verified with the use of Oracle's interface.

6.5.4 Test 8: The Amount of Data Tested by MORD Can Be Varied

To determine the maximum size of database that a machine can adequately support, it is necessary for a benchmark to test databases of different sizes. One method of accomplishing this task is to test for the existence of linear scale-up. Linear scale-up here can be defined as a situation where an increase in the data being manipulated produces an equivalent increase in the response time. For example, if the response time was originally 10ms, by doubling the number of items inserted the expected response time then becomes 20ms. Near linear scale-up should occur until a threshold is reached, where the threshold indicates the largest database size that the system can productively support.

Test 8 reflects a hypothesis based on the theory related to linear scale-up to demonstrate that the size of MORD's database can be increased. Hypothesis 4 is given as:

Increasing the Amount of Data Produces Near Linear Scale-up

The amount of data can be varied both in terms of number as well as size of the items to test scale-up. The relevant benchmark parameter is the variable that controls the number of data items for each query. No database parameters are significant.

In Test 8, the response time is measured for each type of operation. The three values that are assigned to the number of items include a base value, a value half that size, and a value double that size. The amount of data is assigned the values: small, medium, and large. The benchmark configurations for Test 8 are given in Table 6-6.

	Datatype	**Operation**	**Database Size**	**Performance Metric**	**# Items**
Benchmark configuration 1	Image, Audio, Video	Insert, select, update, delete	Small, Medium, Large	Response Time	Half
Benchmark configuration 2	Image, Audio, Video	Insert, select, update, delete	Small, Medium, Large	Response Time	Standard
Benchmark configuration 3	Image, Audio, Video	Insert, select, update, delete	Small, Medium, Large	Response Time	Double

Table 6-6 Benchmark configurations for Test 8

The results of Test 8 are expected to demonstrate that the amount of data can be increased and to determine whether or not linear scale-up occurs. Results showing a trend of linear scale-up would provide some evidence that MORD's results are accurate.

6.5.5 Test 9: The Granularity of MORD's Measurement Can Be Varied

The ability to alter the granularity of the timing is required, for example, when it is important to determine which step out of many is slowing done an operation. Test 9 measures the response time of each individual operation during the insertion of image data. Its results are subdivided into four sections, namely: initialising the object (*IOT*), importing data into the database (*IT*), extracting the data properties (*ET*), and generating a thumbnail for images

(*TGT)*. The variables mentioned in parentheses represent the times to complete the various sections and are related by the following formula:

Tot = IOT + IT +ET + TGT

6.6 Simplicity Tests

6.6.1 Design of Simplicity Tests

The ease of using MORD should be apparent through all tests, and as such no testing is performed. It is still necessary to test explicitly whether or not the benchmark's functionality and test data can be extended easily. The ability to extend a benchmark makes it adaptable to future applications, and as a result increases its lifespan.

6.6.2 Test 10: MORD's Test Data Can Be Extended

The aim of Test 10 is to demonstrate that MORD can be extended to include new data, such as document data. This document data includes one Microsoft Word document, one Acrobat Reader document, one Microsoft Excel spreadsheet, and one PowerPoint document. MORD's design requires that several steps are taken to prepare test data, and so these must be carried out for the document data. After these steps have been completed, the new test data is now ready to be used by MORD in Test 11.

6.6.3 Test 11: Functionality Can Be Added to MORD

Test 11 adds extra functionality to MORD to evaluate the handling of documents, using the document table created in Test 9 and the document test data prepared in Test 10. This extra functionality includes the measuring of the response time for inserting and deleting documents. The code is written in a separate file named `DocExperiment.java`. The outline of this code, showing all the methods, is given in Figure 6-6.

```
public class DocExperiments {
      DocExperiments( … ) { … }
      public OracleConnection connect() throws Exception{ … }
      public void insertRTStm( … )  { … }
      public void deleteRTStm( … )  { … }
      public void control ( … )  { … }
}
```

Figure 6-6 Outline of code to include documents

The code has a constructor, a method to connect to Oracle, methods to test the response time for inserting and deleting the documents, and a method that controls which of the other methods are executed. If additional functionality is added, the code can either be run independently by using a main method, or through the file `MultiUser.java` by editing this file to run the new code. To compile and run the code, the commands given in option A (Figure 6-7) are used for the former, while those given in option B are used for the latter.

```
Option A

javac DocumentTest.java
java DocumentTest

Option B

javac MultiUser.java
java MultiUser
```

Figure 6-7 Code to compile and run new Java classes

The expected result of Test 11 is that the new schema, test data and functionality are all working correctly, thus proving that MORD can be extended.

6.7 Summary

MORD is validated by testing if it meets its objectives of relevancy, adaptability (portability and scalability), and simplicity. Relevancy tests consist of six different tests. These include tests to show that MORD's measurements are repeatable, the timing code is accurate, specific operations can be evaluated, the distribution of the workload can be varied, selected datatypes can be focused on, and the granularity of what is measured can be altered. The relevancy experiment also tests two hypotheses to determine the effect of turning off the caching and logging when bulk loading and the relationship between the sizes of the buffer cache, the amount of repetition in the data, and the performance.

The adaptability experiment checks that MORD is as portable as possible and that it is scalable. The portability tests consist of two tests: the first demonstrates that MORD can connect to different databases while the second shows that the design of the database can be modified. The former also tests the performance of Oracle related to the size of its data block.

The simplicity experiment is also performed in two tests, which demonstrate how additional test data, as well as functionality can be added to extend MORD. The scalability tests consist of a single test to prove that the amount of data, in both size and number of items, can be varied. This also tests whether linear scale-up is achievable in Oracle for multimedia data.

The environment in which these tests are performed does not contain any special physical storage hardware, although it does have a relatively large amount of memory and a relatively fast processor. The default database configuration for the experiments has larger values for `BUFFER_CACHE_SIZE`, `JAVA_POOL_SIZE`, `SHARED_POOL_SIZE`, and `OPEN_CURSORS` than the default Oracle configuration.

Chapter 7 : Results and Informal Analysis

The previous chapter detailed the design of tests to validate MORD. This chapter presents the results and an informal analysis for these tests. The first set of tests indicate that MORD is relevant by showing that its measurements are repeatable, its timing methods are designed correctly, and that both the queries and dataset can be altered. The second set of tests indicate that MORD is adaptable by showing that it can connect to different databases, the design of the databases can be altered, that it is scalable, and that the granularity of the measurements can be altered. The last set of tests indicates that MORD is simple to extend in terms of functionality as well as test data.

7.1 Format of Results

Results are variably presented in the form of bar graphs (Tests 1, 3, 6, 8, 10, and 11), scatter graphs (Test 1), pie charts (Tests 5 and 9), and tables (Tests 5 and 6). Tables of the full results can be found in Appendix C.

MORD can be used to measure either throughput or response time, but for the set of validation tests only the response time is recorded. The response time is measured in milliseconds, and can be presented as total times (usually given in seconds), the arithmetic mean, a percentage, or as the values calculated with additional formulas. The total time is the sum of response times for a set of tests, and thus it is calculated for the number of items, such as 20 images, multiplied by the number of times the experiment is run, such as 5 runs. The total response time is used in Test 1 for the scatter graph of raw results, and in Test 8 as it shows the increase in the total time as the number of items or the database size increases. The average response time is the time taken for a single item and is calculated using the arithmetic mean. It is used in Test 8 to determine if linear scale-up has been achieved.

Since MORD is primarily designed to be a comparative tool, it is useful to present the results using a measurement that is not dependant on the total time, such as percentage. For example, in Test 3 and 6 the results are presented as percentage to show the increase in performance when caching/logging are turned off and with a large block size. Further calculation is sometimes necessary; for example, in Test 1 the standard deviation is required.

The results are recorded for different data formats, and can be presented either separately or as the average of the formats for a datatype. The results are given for each format in Test 1 because it evaluates raw results, and in Test 9 as it shows the variation in the distribution of time between the different formats. Most of the other tests give the average for all of the formats.

The tests are usually run for all datatypes, that is, images, audio, and video, and for all the operations, that is, insert, select, update, and delete, unless explicitly stated. Since the results are generally similar for the various queries and datatypes, only a selection of the results are presented in this chapter. The full set of results can be seen in the Excel timesheets included with the thesis.

7.2 Relevancy Tests

7.2.1 Test 1: Are MORD's Measurements Repeatable

Test 1 indicates whether or not the measurements recorded by MORD are repeatable by examining the variation between the results of consecutive runs of the benchmark. A scatter graph is a useful visual aid to display the data distribution. The response times for inserting and updating a selection of small and medium images are given in Figure 7-1.

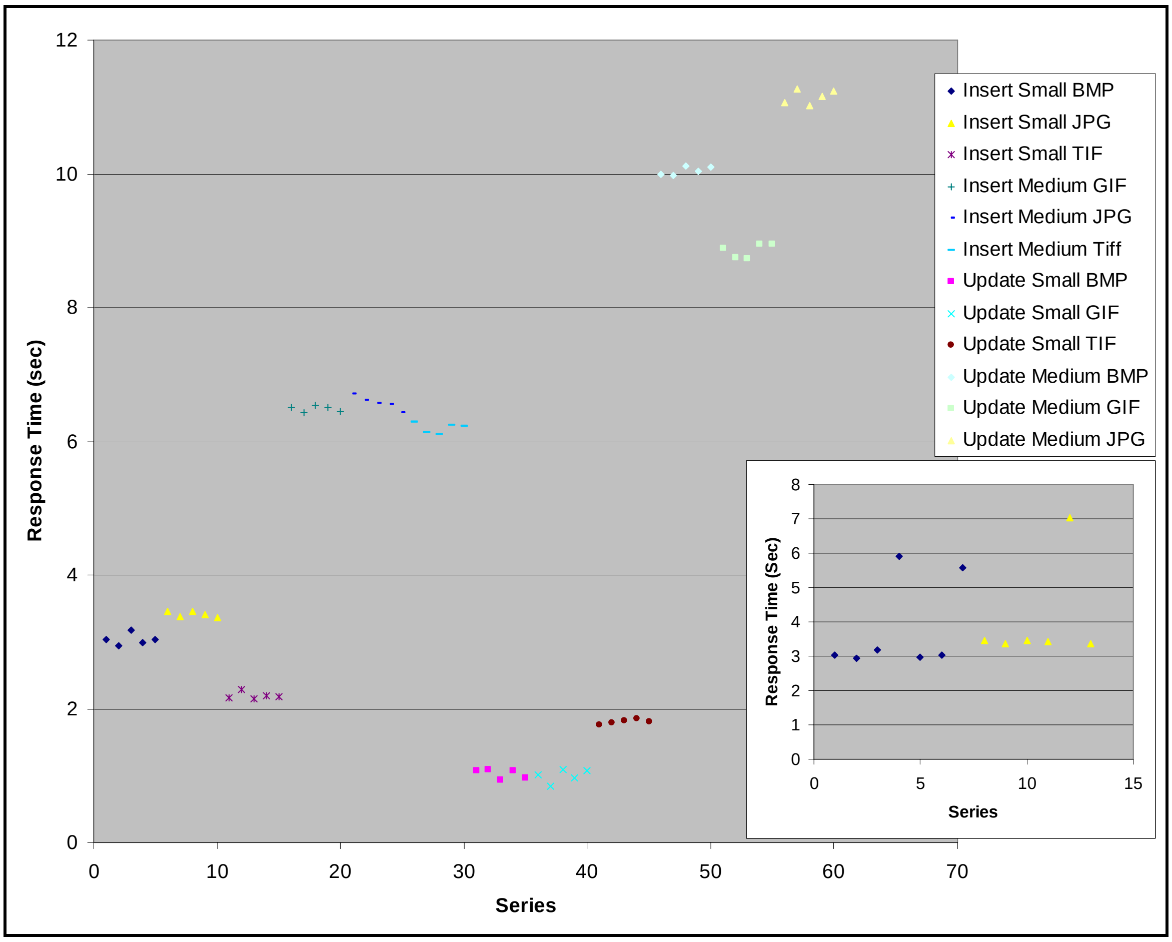


Figure 7-1 Distribution of image response times

The inset shows the results for inserting small BMP and JPG images, where some of the results deviate from the rest of the results. As with all the results, the response times that vary significantly from the other results are excluded. These values are excluded as they are caused by other influences, such as Oracle background processes, and using them would distort the results that are intended to be measured. In the inset graph it can clearly be seen that two results for BMP images and only a single result for JPG images vary considerably from the rest. In the main graph these values have been eliminated in the final results. A similar process was used for all the results.

The patterns seen in Figure 7-1 show that for each set of results the response times are close together, indicating that MORD's measurements are repeatable. The percentage by which the results vary on average is calculated using the formula:

*(Average Standard Deviation/Arithmetic Mean)*100*

The average standard deviation is a measure of the distribution of the response times. It is divided by the arithmetic mean and converted to a percentage so that it is not dependant on the size of the response time. Evidence of repeatability comes from small values resulting from this calculation. Figure 7-2 gives the averages for each datatype/operation.

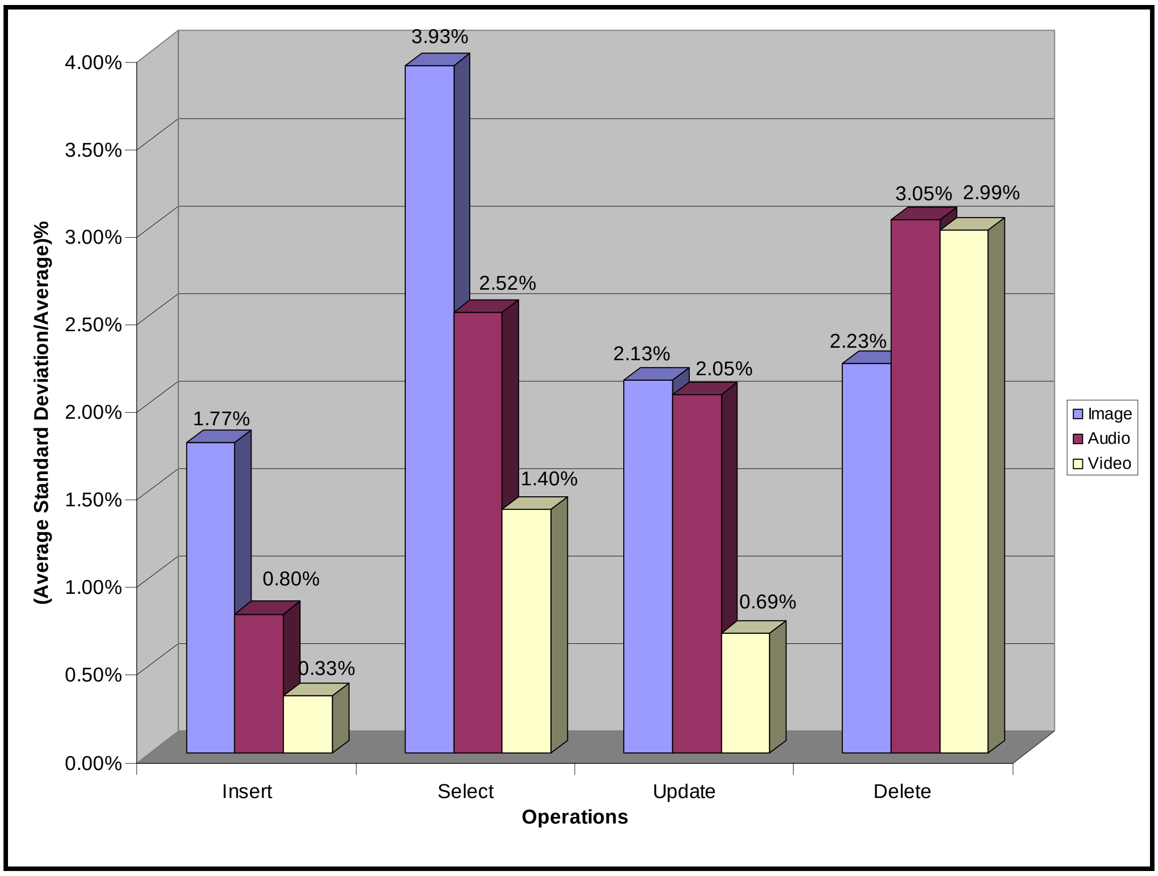


Figure 7-2 Summary of variation in the distribution of the results (Test 1)

The largest resulting percentages that are calculated for the three datatypes, namely image, audio, and video, are all between 5% and 6%. Approximately 73% of the results vary by less than 3%, and 86% vary by less than 4% showing that the amount of variation is small. An informal analysis of these results indicates that MORD's measurements are repeatable, since the values are reasonably small.

7.2.2 Test 2: Is MORD's Method of Timing Designed Correctly

Test 2 checks if MORD's response time method is designed correctly through testing that the time recorded for each individual loop is approximately equivalent to the time recorded for the whole loop. This is achieved by running the tests with additional code to measure the time for each individual operation, as shown in bold in Figure 7-3.

```
ts3 = System.currentTimeMillis();
for (int i=1; i <=num; i++)
{
      ts5 = System.currentTimeMillis();
      …
      ts6 = System.currentTimeMillis();
      timer1 = new String(Long.toString(ts6-ts5));
      num1 = new String(Integer.toString(num));
      text = "Time to initialise 1 audio object is:" + timer1 + "\n";
      f.write(text);
}//for
ts4 = System.currentTimeMillis();
timer2 = new String(Long.toString(ts4-ts3));
num1 = new String(Integer.toString(num));
text = "Time for " + num1 + " audio objects is:" + timer1 + "\n";
f.write(text);
```

Figure 7-3 Timing of individual queries

The times taken for each individual operation, as measured by timer1, are added up, and the sum of these times is compared to the total time returned by timer2.The difference between these is calculated using the formula:

((results for timer2) - Σ (results for timer1))/ (results for timer2)

The resulting differences for audio data in percentages are shown in Figure 7-4.

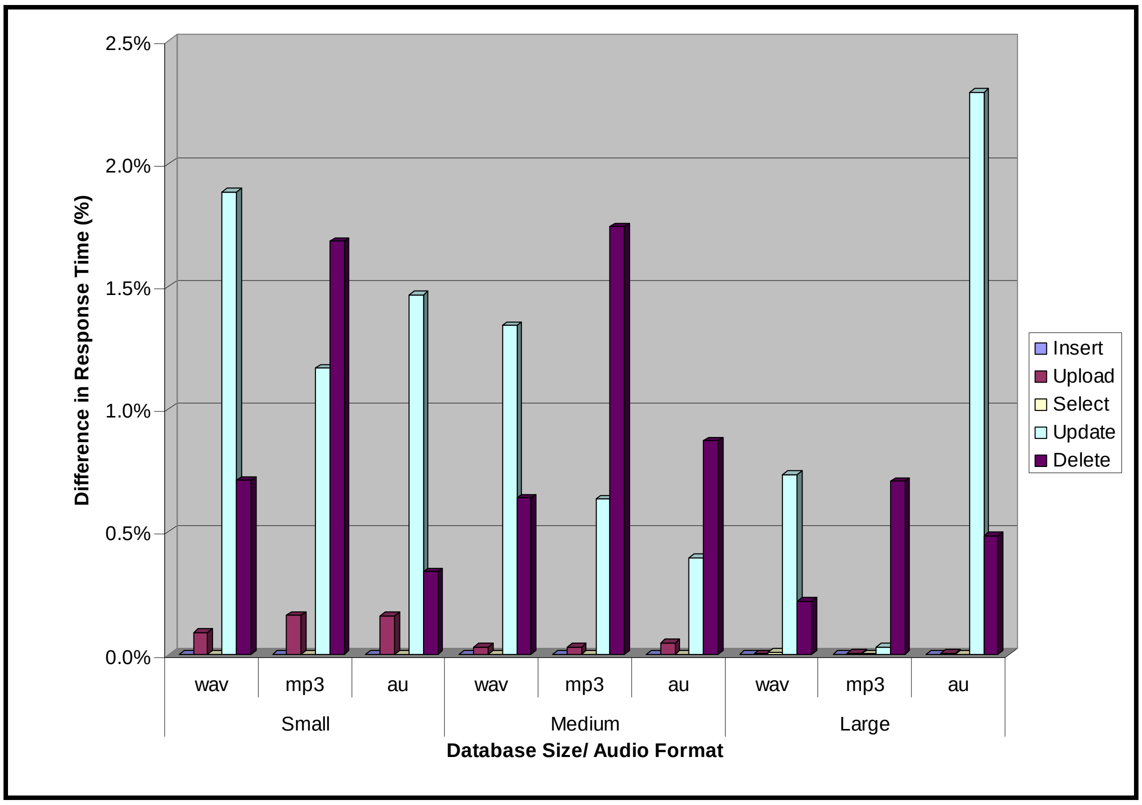


Figure 7-4 Difference between timing individual queries and whole loop (Test 2)

The difference between timing each individual query and the whole loop is small in all cases. The initialisation of an object, uploading of data, and selecting of that data produce insignificant differences, while for the other operations this difference is slightly more noticeable. It can be seen from Figure 7-4 that it never exceeds much more than 2%, and from further analysis it is calculated that approximately 75% of the values have less than a 1% difference. An informal analysis of these results suggests that it is sufficient to measure the whole loop for the response time code.

7.2.3 Test 3: Can MORD Focus on a Query

Test 3 indicates that MORD allows the user to select the queries tested and as a secondary aim it determines the effect on the performance of using caching and logging when inserting data. In the design phase tests have been conducted to show that all of the queries work correctly when selected. To prevent a query from running it must be commented out in the code, as shown for select and update in Figure 7-5.

```
public void control (String cmd)
  {
    OracleConnection con = null;
    String smviddir, medviddir, larviddir, command;

    try
      {
            …
      if(command=="all")
            {
                  callInsert(outData, con);
                  //callSelect(outData, con);
                  //callUpdate(outData, con);
                  callDelete(outData, con);
            }
      …
 }
```

Figure 7-5 Commented out queries

Although it is possible to comment out any of the queries, to get accurate results MORD requires both the insert and delete queries to be performed for every experiment run. If the insert query is not run, there is no data for the other queries, whereas if the delete query is not run, there is too much data, and so excluding either of these leads to incorrect results. The user can therefore only decide whether the select or update queries are performed.

Test 3 was run with the insert and delete operation for two different database configurations, one with caching and logging and the other without these. The results are used to evaluate Hypothesis 1 and are shown in Figure 7-6.

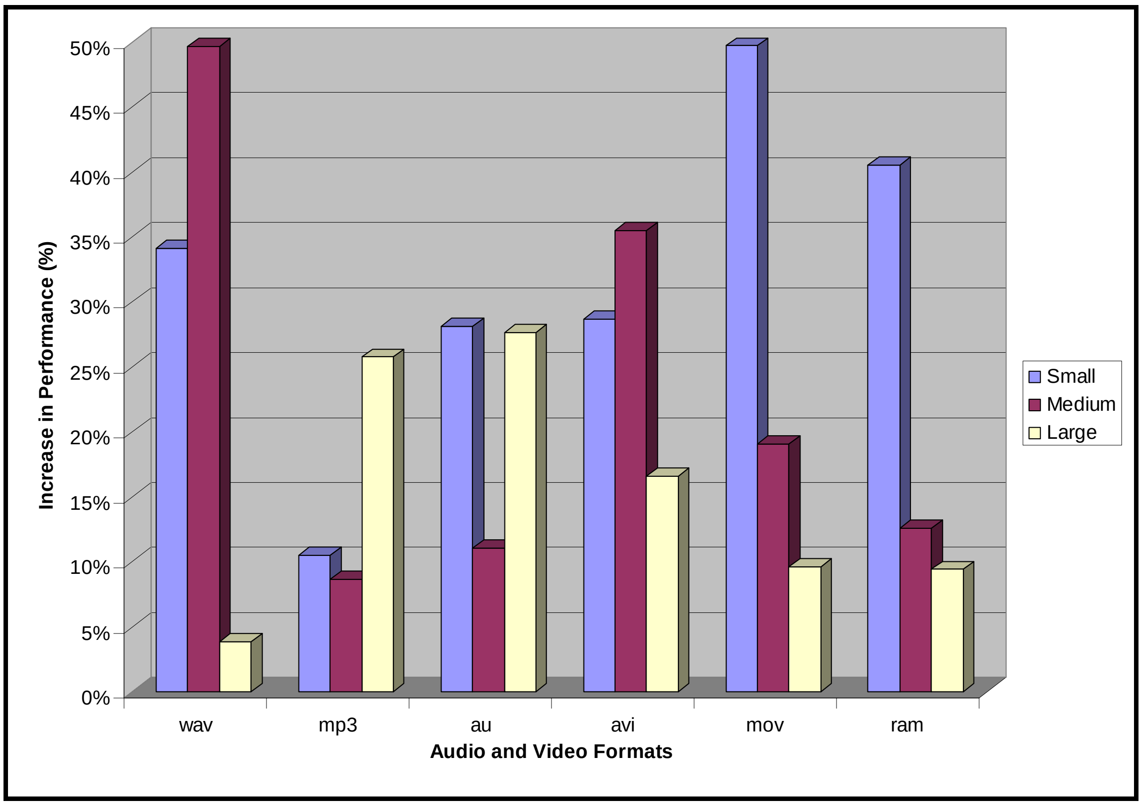


Figure 7-6 Increase in performance when caching and logging are turned off (Test 3)

The informal analysis of these results shows that in all cases for audio and video data there is an improvement in the performance when caching and logging are turned off. Similar results were found for images. The improvement ranged from just less than 5% to nearly as much as 50%, which is significant. Further analysis is required to determine the reasons for this variation in the results, but from these results it is strongly recommended that caching and logging are turned off for bulk loading.

7.2.4 Test 4 and 5: Can MORD's Dataset Be Altered

Test 4 investigates if the composition of MORD's dataset can be altered to contain a single datatype, while Test 5 investigates if the repetition in MORD's dataset can be altered. These are both shown by testing Hypothesis 2, which examines the effect of altering the size of the buffer cache in conjunction with varying the amount of repetition in the dataset for image data.

In these tests it is important that the buffer cache is totally empty prior to running them and, unlike the other tests, the different database sizes must be tested separately. The reason for running these tests in such a manner is that the expected results are based on the size of the dataset. If testing of the different database sizes are mixed, then the amount of data in the buffer cache will change.

It was expected from prior knowledge of Oracle that the results would show a pattern of increasing response times for a decrease in the buffer cache size if the size of the unique data in the dataset exceeds the size of the buffer cache. The size of the unique data is dependant on the amount of data repeated in each of the three datasets, and is as shown in Table 7-1.

Database Size	Small	Medium	Large
Dataset 1	12.5MB	125.0MB	1250.0MB
Dataset 2	3.0MB	30.0MB	300.0MB
Dataset 3	0.2MB	2.0MB	20.0MB

Table 7-1 Total size of image data (5 executions)

Dataset 1 always has the largest amount of unique data as it consists of only distinctive item and thus it is the sum of all of the items. Dataset 2 is much smaller as it only accesses unique data for a quarter of the test, while Dataset 3 is the smallest as it accesses a single data item and so is the size of that item.

There should be no effect on the performance if the dataset's size is smaller than the buffer cache's size, otherwise the response time should be larger to varying degrees depending on the difference between the sizes of the two. Based on this, the expectations are as summarised in Table 7-2.

Buffer Cache Size (MB)	Dataset Type	Response Time for Small Dataset	Response Time for Medium Dataset	Response Time for Large Dataset
16	**1**	No Effect	Larger Response Time	Larger Response Time
	2	No Effect	Larger Response Time	Larger Response Time
	3	No Effect	No Effect	Larger Response Time
100	**1**	No Effect	Larger Response Time	Larger Response Time
	2	No Effect	No Effect	Larger Response Time
	3	No Effect	No Effect	No Effect
320	**1**	No Effect	No Effect	Larger Response Time
	2	No Effect	No Effect	No Effect
	3	No Effect	No Effect	No Effect

Table 7-2 Expected results for Hypothesis 2

This means, for example, that the results for the database configuration with a buffer cache of 320MB should be approximately the same in all cases, with the exception of Dataset 1 with large data which should have a larger response time. The actual results are given in Table 7-3.

Buffer Cache Size (MB)	Dataset Type	Response Time for Small Dataset (ms)	Response Time for Medium Dataset (ms)	Response Time for Large Dataset (ms)
16	1	7.98	8.95	9.42
	2	7.60	8.23	8.38
	3	7.47	7.55	7.90
100	1	7.96	8.49	8.86
	2	7.69	7.72	8.03
	3	7.29	7.44	7.47
320	1	7.80	7.98	8.63
	2	7.52	7.62	7.60
	3	7.30	7.37	7.41

Table 7-3 Actual results for Hypothesis 2

These results show a general trend, where the entries that were marked as not being affected are all smaller than 8ms and those that were marked as having a larger response times are all greater than 8ms. It is also interesting to note that the largest response time was produced for the database with the smallest buffer cache and the largest dataset. As expected, an informal analysis of these results suggests that the performance is slower if the dataset is larger than the buffer cache.

Ideally, the response times should be the same for the tests where the dataset fits into the buffer cache, but when working with such small values it is difficult to get precise measurements. Any small variation that is caused, for example, by a background process, could have easily altered the results by a fraction of a millisecond.

A better understanding of the effect that altering the size of the buffer cache and the repetition in the dataset has on the performance can be gained by calculating the percentage by which the response time improves. This is achieved for Dataset 1 and Dataset 2 using the formula:

$$((DS1 - DS2)/DS1)*100$$

Where DS1 = Dataset 1

DS2 = Dataset 2

A similar formula is used for Dataset 2 and Dataset 3. The results from applying this formula are shown in Figure 7-7.

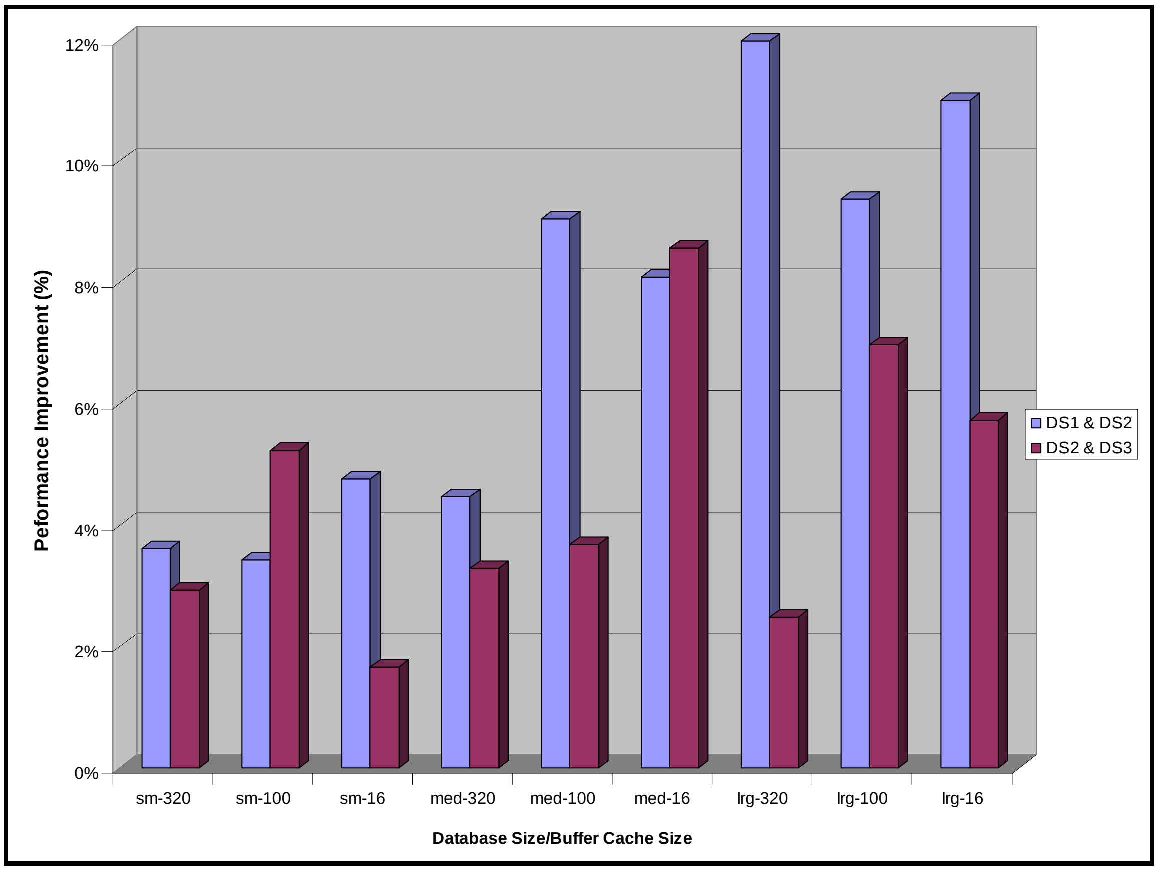


Figure 7-7 Improvement in performance related to buffer cache and dataset

An informal analysis of these results shows that the combination of buffer cache size and dataset affects the results of the smaller database less than it does for the larger database, as expected. It also shows that when the size of the buffer cache is larger, the effect is smaller.

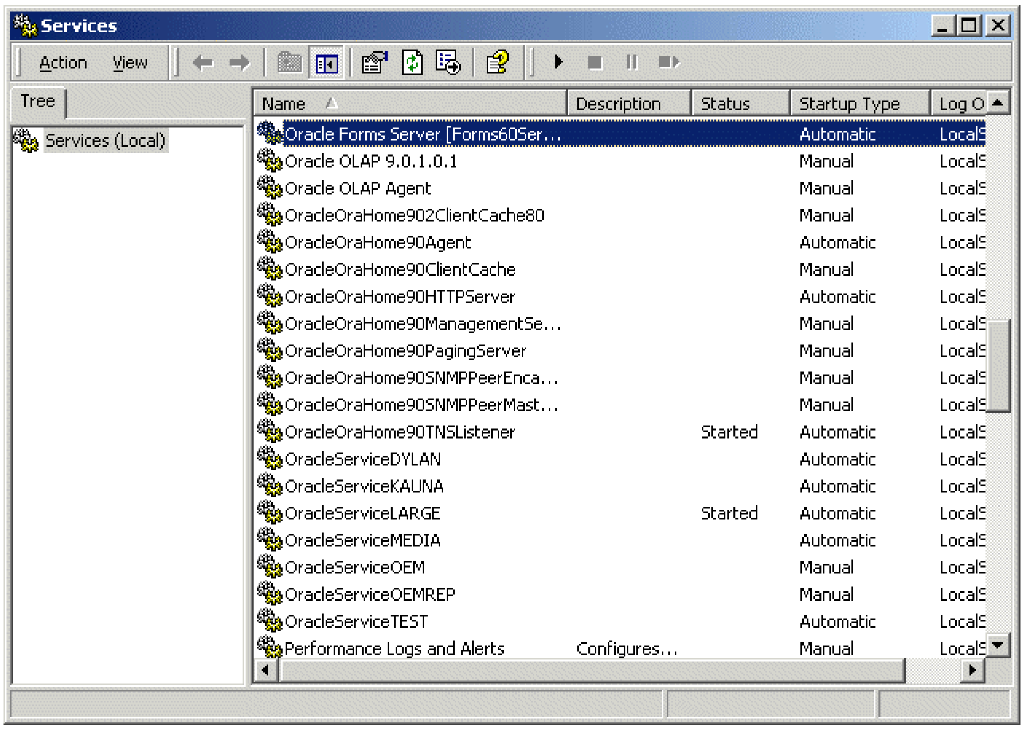
7.3 Adaptability Tests

7.3.1 Test 6: Can MORD Connect to a Different Database

Test 6 aims to show that MORD can connect to different databases by testing the performance of a static parameter. A static parameter requires a different database for each value tested, which in this case is the one that controls the data block size.

It is important to ensure that MORD is being applied to the correct database, especially if there is more than one database being tested. This has been established prior to testing throughout the experiments, but it is explicitly discussed in this test. In Test 6 it is crucial to test that the correct database is being evaluated for each execution of MORD as it tests more than one database.

The easiest way to verify that a certain database is being accessed is to shutdown the other databases, and make sure that MORD still runs without producing error messages. Shutting down a database can be performed through the Oracle Enterprise Manager Console or by shutting down the database service for that database. The screenshot in Figure 7-8 shows the Windows 2000 interface to perform the latter.



Name	Description	Status	Startup Type	Log O
Oracle Forms Server [Forms60Ser...			Automatic	LocalS
Oracle OLAP 9.0.1.0.1			Manual	LocalS
Oracle OLAP Agent			Manual	LocalS
OracleOraHome902ClientCache80			Manual	LocalS
OracleOraHome90Agent			Automatic	LocalS
OracleOraHome90ClientCache			Manual	LocalS
OracleOraHome90HTTPServer			Automatic	LocalS
OracleOraHome90ManagementSe...			Manual	LocalS
OracleOraHome90PagingServer			Manual	LocalS
OracleOraHome90SNMPPeerEnca...			Manual	LocalS
OracleOraHome90SNMPPeerMast...			Manual	LocalS
OracleOraHome90TNSListener		Started	Automatic	LocalS
OracleServiceDYLAN			Automatic	LocalS
OracleServiceKAUNA			Automatic	LocalS
OracleServiceLARGE		Started	Automatic	LocalS
OracleServiceMEDIA			Automatic	LocalS
OracleServiceOEM			Manual	LocalS
OracleServiceOEMREP			Manual	LocalS
OracleServiceTEST			Automatic	LocalS
Performance Logs and Alerts	Configures...		Manual	LocalS

Figure 7-8 Screenshot of running services

This screenshot displays a list of the services running during an execution of MORD, where in this particular case the LARGE database service is the only database open. It can be presumed that different databases are being evaluated by MORD if the results for Test 6 show a noticeable variation between the two database configurations. These results also assist in determining the effect that `BLOCK_SIZE` has on the performance.

To calculate the improvement in the performance, further calculations must be performed on the results using the following formula:

$$(\Sigma(\textit{results for 4K}) - \Sigma(\textit{results for 8K}))/\Sigma(\textit{results for 8K})$$

This formula calculates how much longer it takes, in percentage, when using a value of 4k for the `BLOCK_SIZE` as opposed to a value of 8k. The increase in the performance as a result of using a larger block size can be seen for audio data in Figure 7-9.

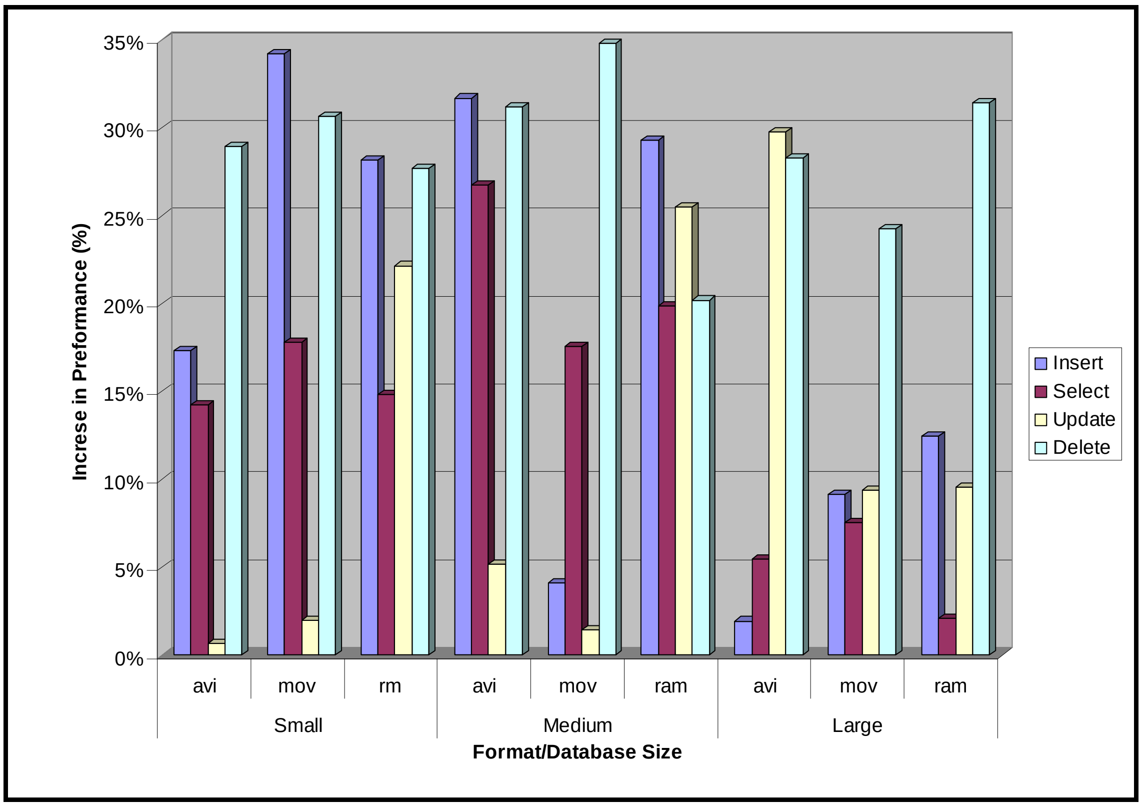


Figure 7-9 Increase in performance if the block size is doubled (Test 6)

These results show that the response time decreased in all case for the queries performed on the audio data when the database had a larger block size, 8k `BLOCK_SIZE`. The increase in performance ranges from less than 5% to just more than 30%. Similar results were found for the other datatypes.

A summary of the results for all the datatypes, images, audio, and video, and all the query types, insert, update, select, and delete, are shown in Table 7-4.

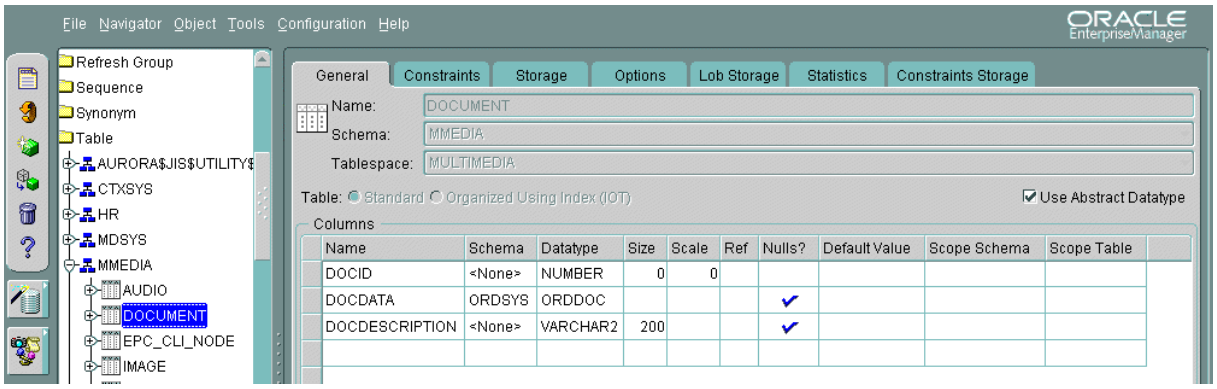
Datatype	% increase in response time
Images	11%
Audio	11%
Video	16%

Table 7-4 Summary of performance improvement for all datatypes

An informal analysis of the results shows an improvement at least 11% in all cases. These results confirm that MORD has evaluated different databases and suggest that a larger `BLOCK_SIZE` improves the performance of the database. Although having a larger `BLOCK_SIZE` does come at the cost of possible space inefficiency, multimedia data typically requires large amounts of space and thus wasting such small amounts is usually inconsequential. It is implied from these results that, a larger `BLOCK_SIZE` improves the performance when dealing with multimedia data, as holds with standard data.

7.3.2 Test 7: Can the Design of the Database be altered with MORD

Test 7 indicates that the design of MORD's databases can be altered by adding a table to store documents. The new table is created, as done with all the other tables, by running the schema script, which has been altered to include the Document table. After running this script, the Oracle Enterprise Manager can be used to check that the table has been created correctly. In the screenshot shown in Figure 7-10 the details of the `DOCUMENT` table can be seen.



Name: DOCUMENT
Schema: MMEDIA
Tablespace: MULTIMEDIA
Table: [✓ Standard] Organized Using Index (IOT) [✓ Use Abstract Datatype]
Columns

Name	Schema	Datatype	Size	Scale	Ref	Nulls?	Default Value	Scope Schema	Scope Table
DOCID	<None>	NUMBER	0	0					
DOCDATA	ORDSYS	ORDDOC				✓			
DOCDESCRIPTION	<None>	VARCHAR2	200			✓			

Figure 7-10 Screenshot to illustrate that the document table has been created (Test 7)

The `DOCUMENT` table consists of three columns to store the `ID`, the actual documents, and a description of the documents, as it is supposed to.

7.3.3 Test 8: Is MORD Scalable

Test 8 indicates that MORD allows the user to vary the number of data items manipulated, as well as test if linear scale-up occurs. When graphing the total times for different query/datatype combinations, the fact that the number of items has increased is evident by the increase in total response time. Three datasets are used where Dataset 2 consists of double the number of items in Dataset 1 and half the number of items in Dataset 3. These datasets are also selected so that their sizes increase by the same proportion.

It is easy to verify that the number of data items manipulated has changed, using either the SQL query `COUNT` or the SQL query `SELECT` and manually counting, whereas it is more difficult to verify that MORD correctly measured the response time. One method of testing this is to check if the results are as expected, where the response times for Dataset 1 should be approximately half the values recorded for Dataset 2, and in turn these values should be approximately half the size of the values for Dataset 3. This can be seen in Figure 7-11 where the total response times for inserting and updating are presented.

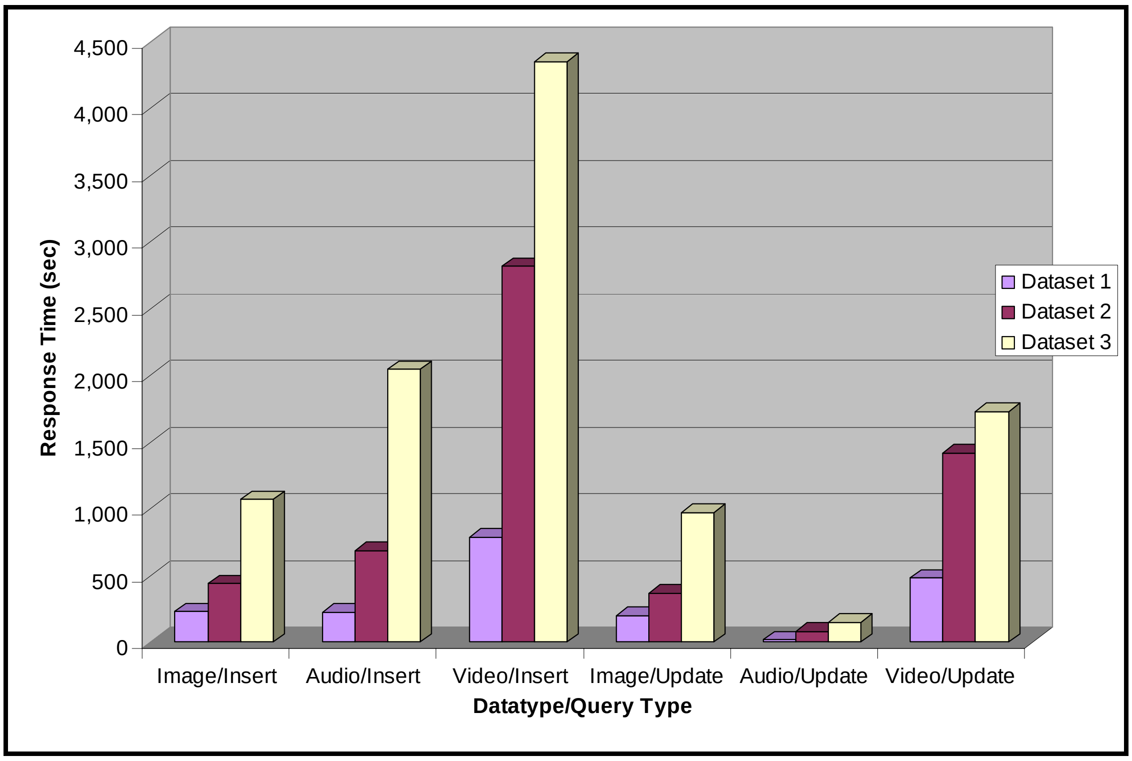


Figure 7-11 Effect of altering the number of items (Test 8)

The results are as expected, as can be illustrated using the results for inserting images. Here, Dataset 1 produces a response time of 227 seconds, Dataset 2 produces a response time of 439 seconds (which is almost double the first amount), and Dataset 3 produces the response time 1,070 seconds (which is just over four times the first amount). Since there is an increase in the total response times this suggests that MORD is correctly measuring the response time as the number of items in the dataset is increased.

Further analysis is still required to determine if linear scale-up has occurred. The average results need to be normalised by dividing the total response time with the number of items for each. The resulting values can be used to determine the effect that altering the number of items has on the performance, where linear scale-up exists if the average results for a query type are the same. The results for selecting and deleting images are used to illustrate this, as seen in Figure 7-12, where both the total response times and their average values are presented.

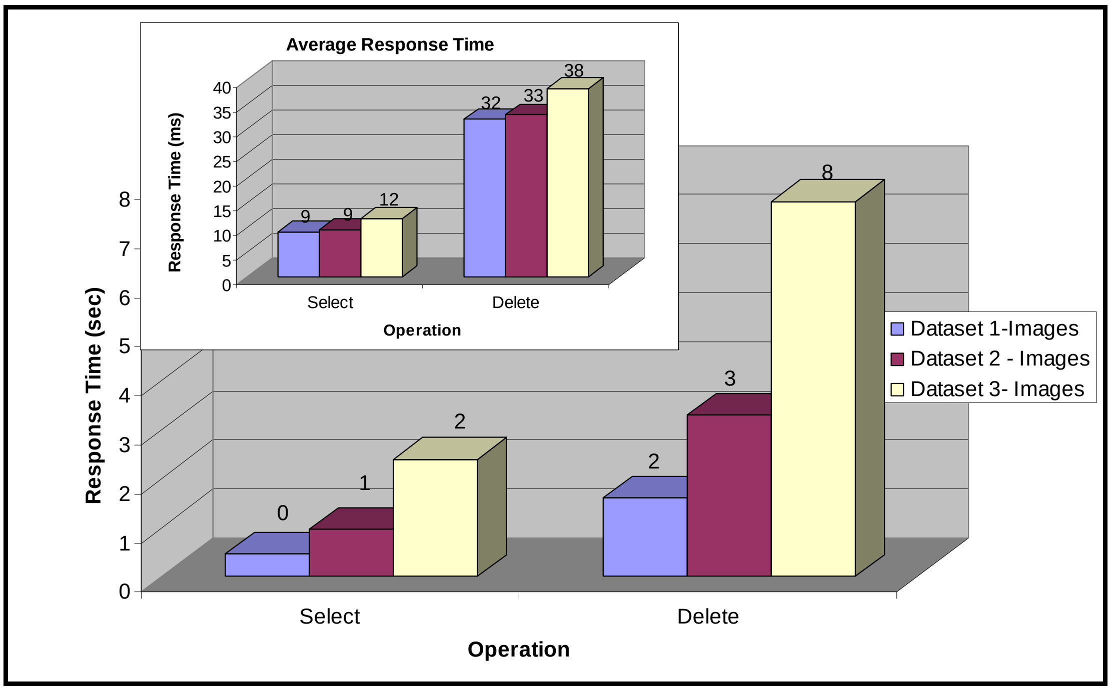


Figure 7-12 Effect of altering the number of images for selecting and deleting (Test 8)

When comparing the results for Dataset 1 and Dataset 2, the response times for the select queries differ by less than 5% and for the delete queries differ by approximately 3%. These values are small, showing that in both cases near linear scale-up is achieved. If the variation between Dataset 2 and Dataset 3 are compared there is a difference of 25% for the select query and 16% for the delete query. This is a much larger variation than found between Dataset 1 and Dataset 2, suggesting that a threshold has been reached somewhere between 20 and 40 images. If the number of items was further increased, it is likely that other thresholds would be identified. Another possible explanation for such results is that there are initial actions that take the same amount for all database sizes, such as initialising the object. If these constant times are, for example, divided by a smaller amount when scaling the results, their resulting values will have a greater effect on the overall results and thus make it seem as though it takes longer for individual tasks.

To test if a database is scalable, an increase in the number of items as well as an increase in the size of the items need to be tested. The three database sizes that are measured are small, medium, and large, where the increase in size is by a factor of 10 for images, and approximately 7 for medium audio and 5 for large audio. Since the databases are scaled by about these factors, a rough method of calculating if linear scale-up exists is to scale these results by dividing the response time for the medium database by the medium database factor and the large database by the medium factor multiplied by the large factor. If linear scale-up has occurred, the results should all be the same for a particular query/datatype combination. Therefore the expected outcome is a pattern where the scaled results are the same for small, medium and large databases. Figure 7-13 shows the results for inserting and updating both audio and image data for the three different sized databases, as well as the scaled results for these.

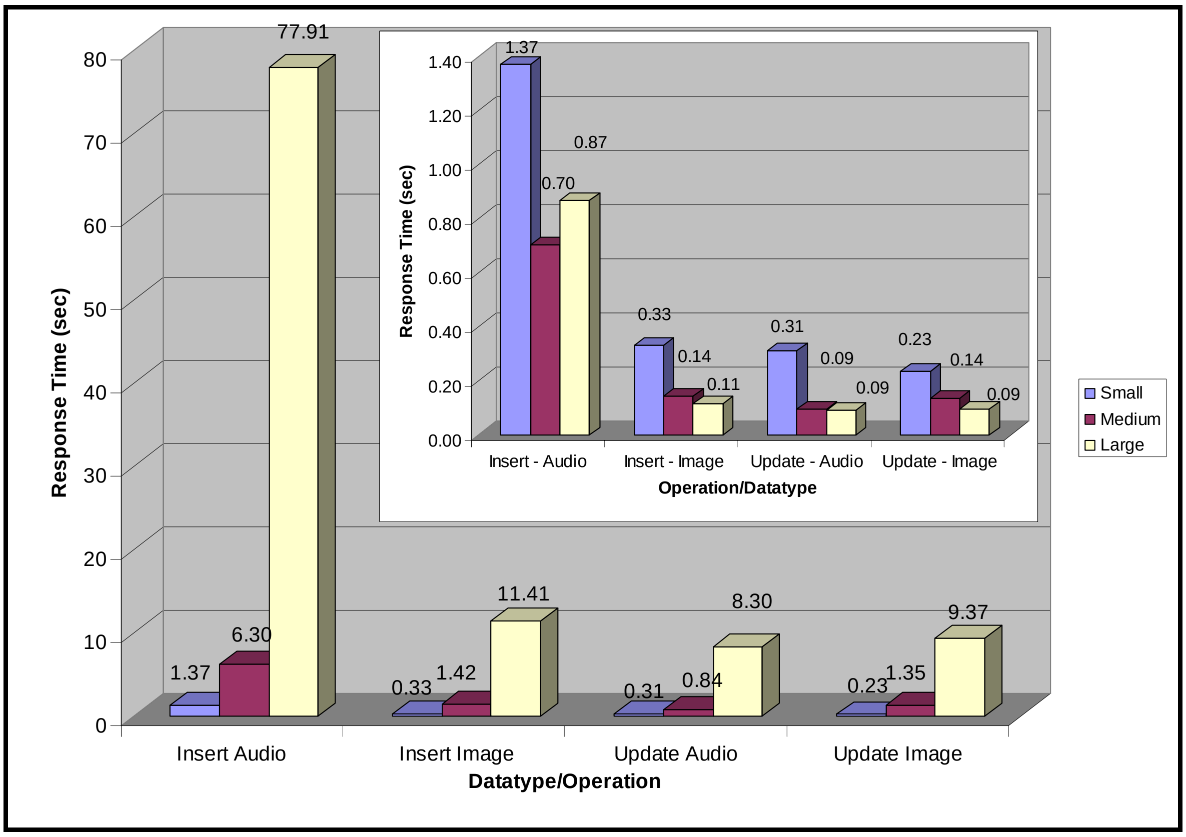


Figure 7-13 Effect of increasing the size of the data (Test 8)

It is evident that the databases have been increased in size from the results presented in the main graph, but it does not appear as if linear scale-up has been achieved from the graph in the inset. Contrary to expectations, an informal analysis of these results implies that there is an improvement in the performance when the size of the data increases. This implies that better than linear scale-up can be achieved.

7.3.4 Test 9: Can the Granularity of MORD's Measurements be Varied

Test 9 investigates if the granularity of MORD's time measurements can be altered, and determines where the time is spent during the insertion of images. The response time is divided into the time to initialise an object, upload data, extract its properties, and generate a thumbnail. (To measure the time to generate a thumbnail requires that extra functionality is added and that the schema is altered to include a column in the table to store thumbnails.) The pie chart in Figure 7-14 gives the break down of the average response time to insert images.

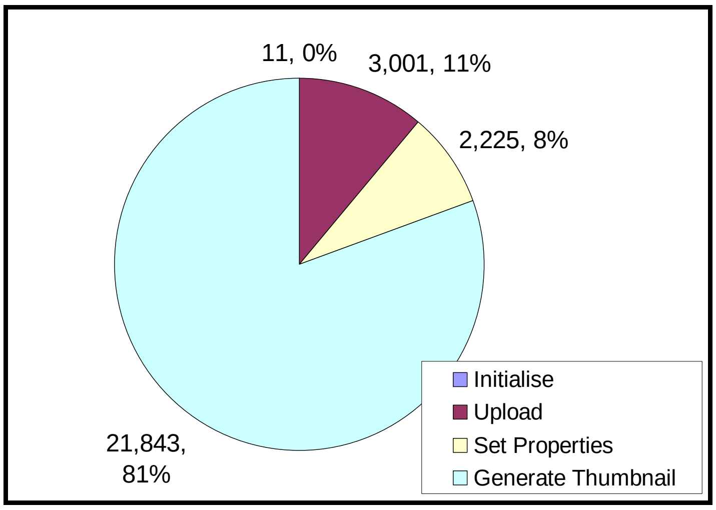


Figure 7-14 Distribution of time during the insertion of images (Test 9)

As expected, an informal analysis of these results shows that the generation of thumbnails takes up a large percentage of time. (Fortunately this task is often not required.) Uploading the data takes the next longest amount of time, while the initialisation of the objects takes the least amount of time.

The results can be further analysed, investigating the performance of different image formats. Five different formats are tested (section 5.3.2.2), BMP, GIF, JPEG, PNG, and TIFF. The results presented in Figure 7-15 show a breakdown of the different formats, with a separate pie for each image's format.

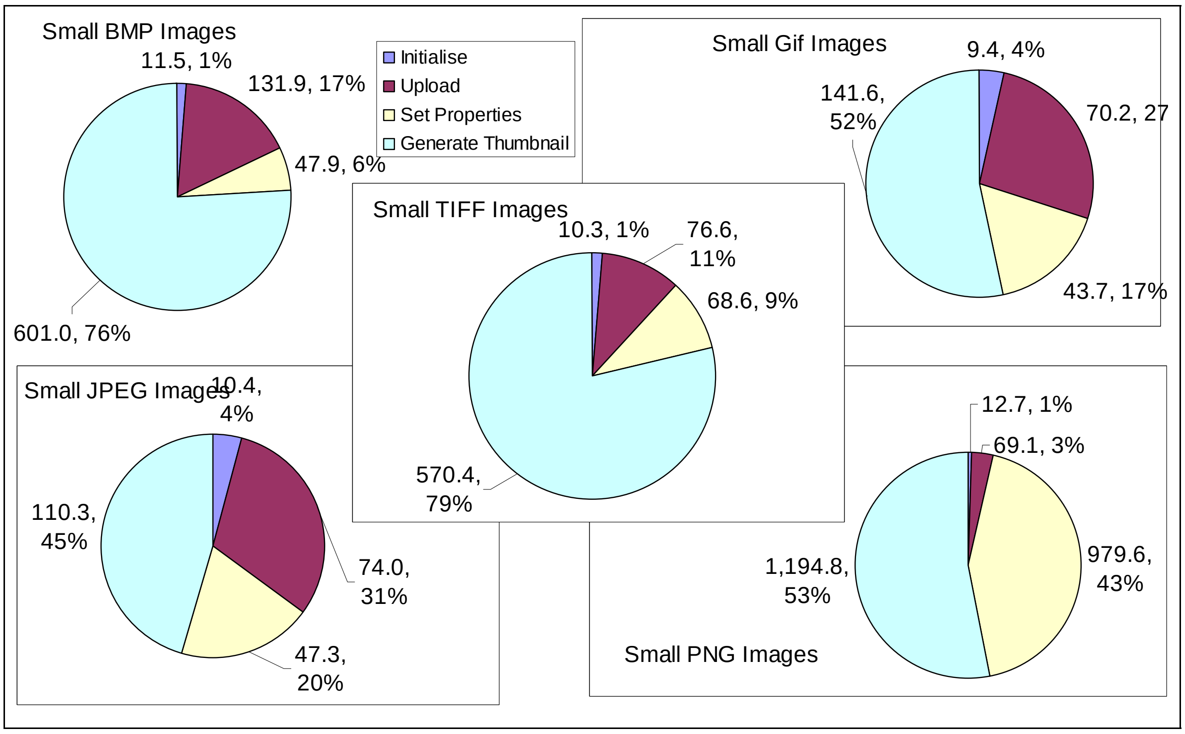


Figure 7-15 Time taken for various tasks of the insert operation with images (Test 9)

These results reveal where the different formats have similar as well as dissimilar response times for the various tasks. It can be seen from an informal analysis of these results that there is a difference of approximately 3ms between the response times to initialise the objects, suggesting that, as expected, this time is not dependant on the format of image. The times to upload an image and to set the properties of an image are also similar among the formats, with the former at around 70ms and the latter in the region of 50ms. Some variation does occur, which suggest that such tasks are influenced by the format of the image. The most noticeable deviation from the response times of the other formats occurs when setting the PNG properties, which takes approximately 20 times longer. There also appears to be considerable variation between the response times to generate thumbnails, suggesting that this task is also dependant on the format of the images.

Examining the results of the different formats separately can be used to explain performance patterns and anomalies that would be otherwise inexplicable. An informal analysis of the results identified one such anomaly is identified in the pie chart in Figure 7-14, where it is surprisingly seen that the time taken to upload an image is only slightly faster than the time to set its properties. Based on intuition, it would be expected that the time taken to extract the image's properties would be much shorter. Another noticeable irregularity is the fact that the total time for PNG images is much longer than that of the other image formats. Both of these irregularities can be explained by the fact that PNG images take a substantial amount of time to read the headers and extract its properties (as identified from Figure 7-15), thus skewing the results. A very different distribution of the time between upload of images and extraction of the properties can be seen from the results of the informal analysis given in Figure 7-16 when the response times for the PNG images have been disregarded.

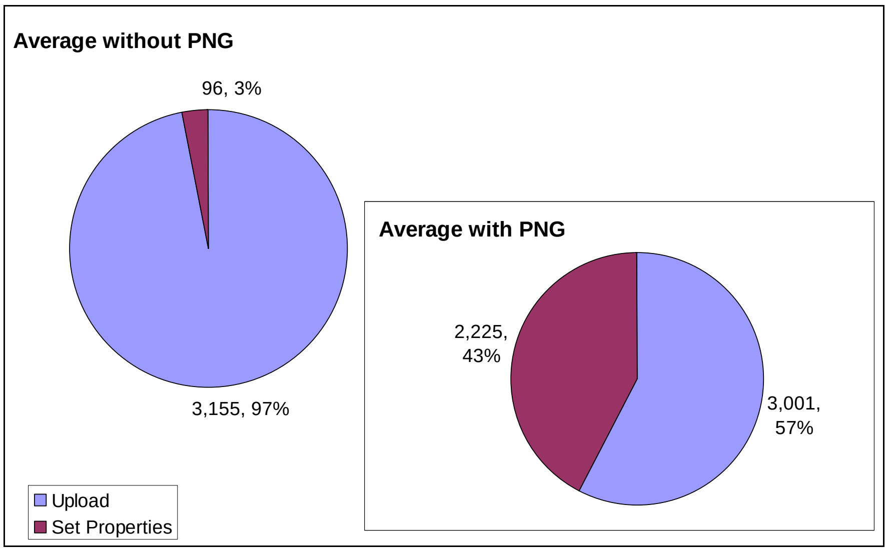


Figure 7-16 Comparison of distribution when including or excluding PNG images (Test 9)

In the pie chart that excludes PNG, the uploading of the data is considerably larger, at 97%, than the extracting of the properties, which is only 3%, whereas the pie chart with PNG shows these values to be similar.

The distribution of the results is not only influenced by the format of the test data but also by its size. This is illustrated in Figure 7-17 where a different pie chart is given for each database size.

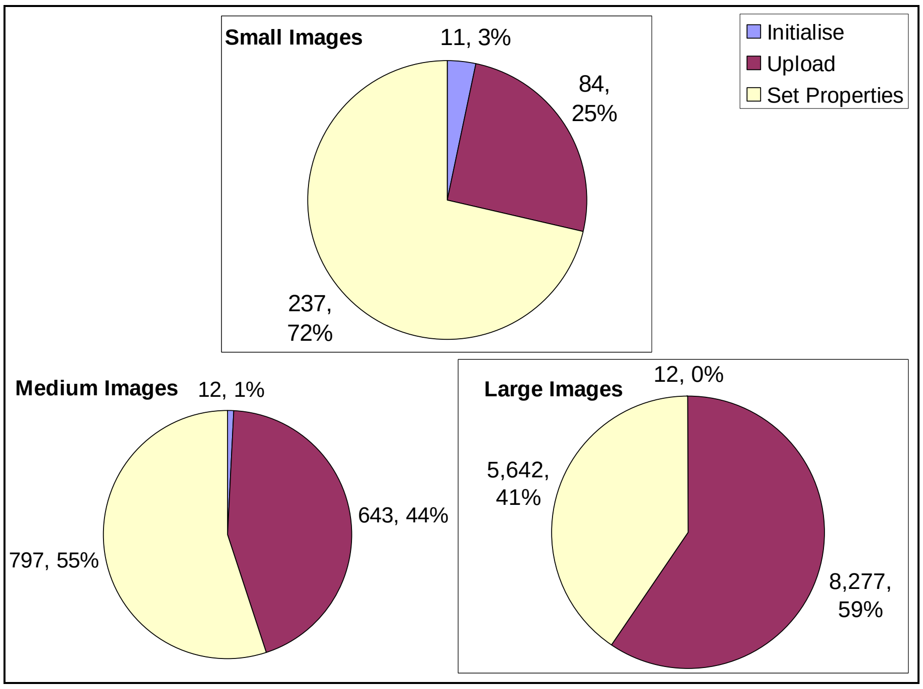


Figure 7-17 Comparison of distribution for different database sizes (Test 9)

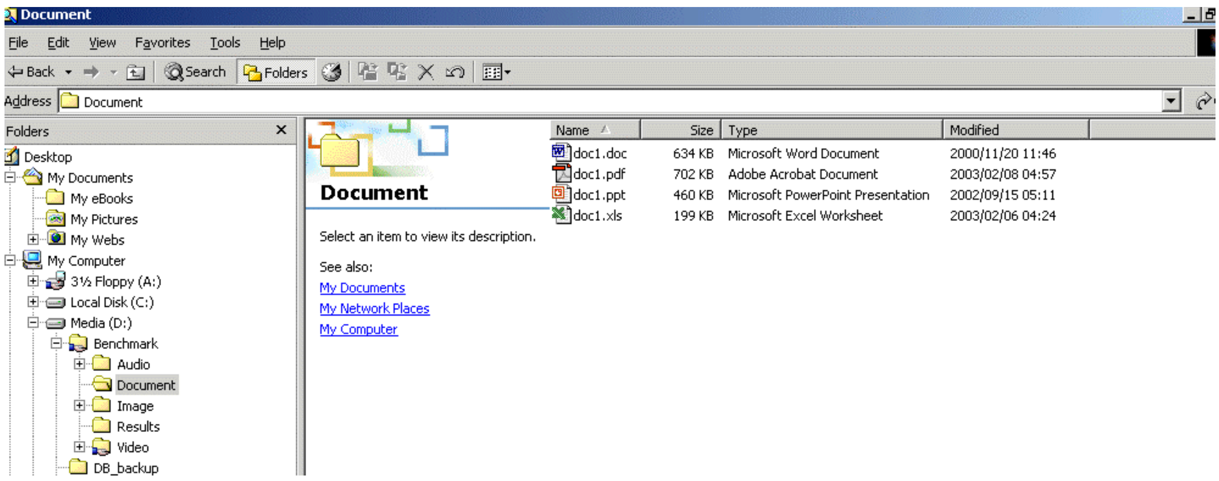
These pie charts suggest that the initialisation of the object is unaffected by the size of the image, whereas the uploading of data is totally dependant on it, as expected. The uploading of the data is 25% for small images, 44% for medium images, and 59% for large images, which produces an overall increase of 34%.

7.4 Simplicity Tests

7.4.1 Test 10: Can Additional Data Easily be Added to MORD

Test 10 investigates if additional test data, such as documents, can be added to MORD. To add new test data, the same steps are taken as those used to setup the original test data. (These are given in Appendix B to setup MORD and again here to explain how new test data is added.)

The first step is to collect the test data, for example the document data for Test 10, and create a directory in which this data is placed, for example the Document directory shown in Figure 7-18.



Name	Size	Type	Modified
doc1.doc	634 KB	Microsoft Word Document	2000/11/20 11:46
doc1.pdf	702 KB	Adobe Acrobat Document	2003/02/08 04:57
doc1.ppt	460 KB	Microsoft PowerPoint Presentation	2002/09/15 05:11
doc1.xls	199 KB	Microsoft Excel Worksheet	2003/02/06 04:24

Figure 7-18 Directory and files for document test data

These files must be renamed using a system that is compatible with the way in which the code to query the data is written. The method that is currently used by MORD is a system where the new data is named with a prefix, for example ‘doc1’, plus the file extension assigned by the OS, such as ‘.ppt’. The document test data consists of data that differs in size as well as format, to produce a more representative sample.

The second step is to verify that the access rights of the data are set to read/write so that they can be uploaded correctly by Oracle. Oracle can only create a link to “read only” data as it does not have access rights to upload it, and although Oracle does not issue a warning message at upload time, subsequent queries, such as extracting the properties, do not work and produce error messages.

The third step is to enable the database to upload data from a directory by creating the directory in Oracle and granting the database access to it. Code must be executed to make this directory known to the database, such as the sample code given in Figure 7-19.

```
create or replace directory ORDDOCDIR as ‘DIRECTORY LOCTATION’;
grant read on directory ORDDOCDIR to public with grant option;
```

Figure 7-19 Code to make Oracle aware of the directory

To check that this has been executed correctly and that Oracle has access to this directory, the `Sys.Dir$` table can be queried. This produces the results shown in Figure 7-20, where the last row is the location of the Document directory.

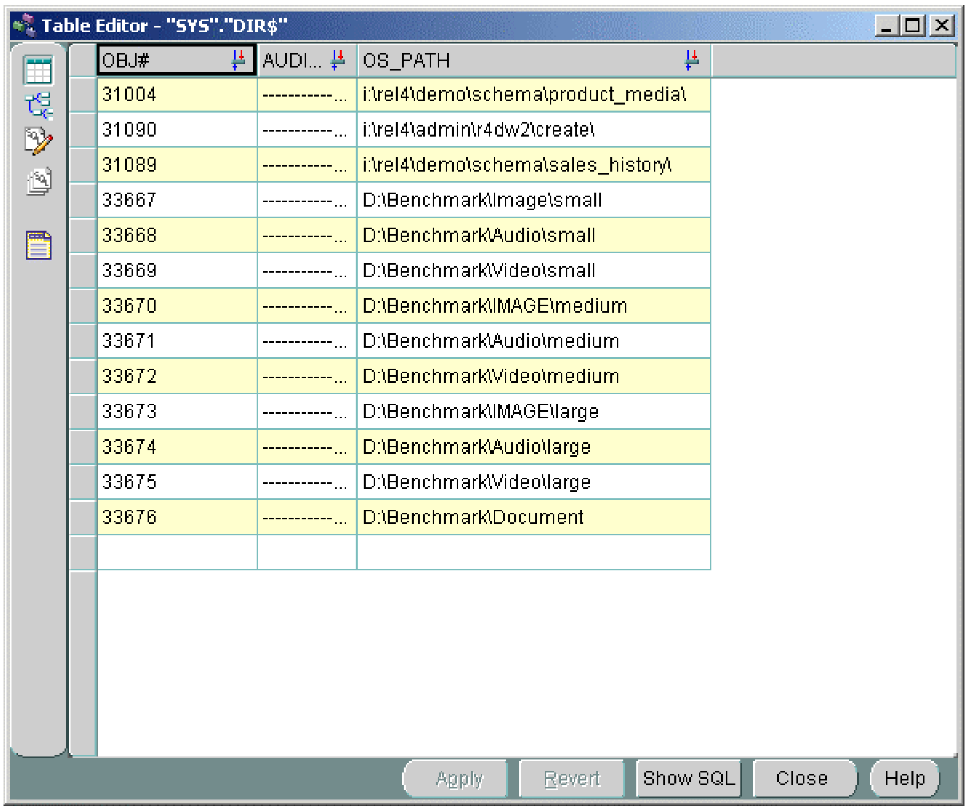


Table Editor - "SYS"."DIR$"

OBJ#	AUDI...	OS_PATH
31004	-----------...	i:\rel4\demo\schema\product_media\
31090	-----------...	i:\rel4\admin\r4dw2\create\
31089	-----------...	i:\rel4\demo\schema\sales_history\
33667	-----------...	D:\Benchmark\Image\small
33668	-----------...	D:\Benchmark\Audio\small
33669	-----------...	D:\Benchmark\Video\small
33670	-----------...	D:\Benchmark\IMAGE\medium
33671	-----------...	D:\Benchmark\Audio\medium
33672	-----------...	D:\Benchmark\Video\medium
33673	-----------...	D:\Benchmark\IMAGE\large
33674	-----------...	D:\Benchmark\Audio\large
33675	-----------...	D:\Benchmark\Video\large
33676	-----------...	D:\Benchmark\Document

Figure 7-20 Query result showing the directories in Oracle

7.4.2 Test 11: Can Additional Functionality Easily be Added to MORD

After Test 10 is complete, Test 11 can be carried out to show if functionality to query such data can also be added to MORD. Tests are run to investigate if test data in Test 10 as well as the functionality in Test 11 have been added to MORD correctly.

It is interesting to collect the response times for the document test data and compare them to the response times for the other datatypes. The number of items used in each test must be reduced to a single item per data format, as there is only one entry for each of the document formats. The previous results have shown that the response time is affected by the size of the multimedia data, therefore, based on this the response times for the documents should be in-between that of image and audio data due to its size. The graph given in Figure 7-21 shows the average response times for inserting data and deleting data.

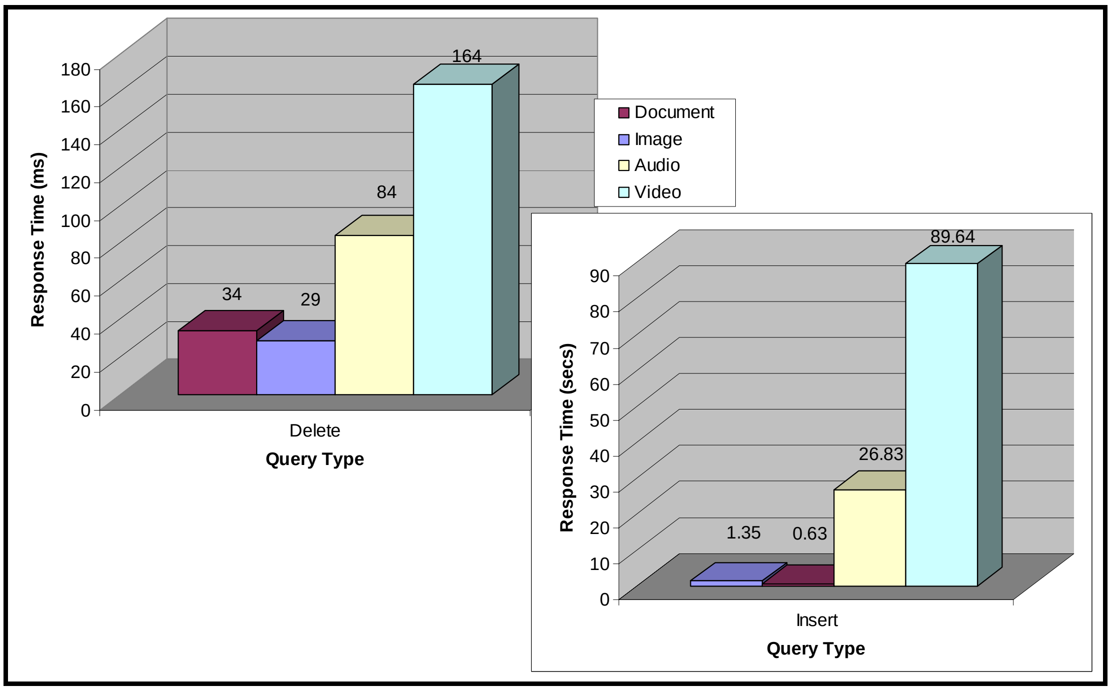


Figure 7-21 Documents results compared with other datatypes results (Test 11)

An informal analysis of the results is as expected, thus suggesting that MORD has been extended correctly.

7.5 Summary of Results

7.5.1 Relevancy Tests

It was indicated that MORD is relevant by performing an informal analysis of the results, demonstrating that MORD's results are repeatable, MORD's designed correctly, the operations and datatypes that MORD tests can be selected, and the amount of data repeated in MORD's dataset can be changed. Test 1 suggests that MORD is repeatable due to significantly small standard deviations and Test 2 demonstrates that the timing methods used by MORD are designed correctly.

Test 3 and 4 showed that MORD can be altered to focus on a specific operation and datatype, while Test 5 demonstrated that the composition of the dataset is adjustable. Test 3 also investigated Hypothesis 1 which states that the speed of insertion is always faster when logging and caching are turned off. Test 4 and Test 5 together showed that increasing the size of the buffer cache improved the performance of the select operations as expected.

7.5.2 Adaptability Tests

It was indicated that MORD is adaptable by demonstrating that it is portable, scalable, and the granularity of its measurements can be altered. Evidence that suggests that it is portable came from Test 6 which showed that it can connect to different databases and Test 7 showed that the design of the database is alterable. The informal analysis of the results of Hypothesis 3 in Test 6 indicated that a shorter response time is produced if using a larger `BLOCK_SIZE` as expected.

The scalability of MORD was supported by the fact that an increase in the number of items or the size of the data also produced an increase in the response time. The informal analysis of these results produced a trend as given in Hypothesis 4 of near linear scale-up.

Verification that the granularity of MORD's measurements can be altered came from Test 9. An informal analysis of the results of this test suggested that the time to upload data is dependant on the size of the data and that the time to initialise an object is minimal.

7.5.3 Simplicity Tests

The simplicity of extending MORD was investigated by the addition of documents to the test data as well as the relevant functionality to measure its performance. Tests 10 and 11 combined tested the performance of documents. The size of the documents was between that of images and audio data, and an informal analysis of the results revealed that the response times were also between these two datatypes. This suggested that the modification was accurate, since the response time is generally related to the size of the data.

7.6 Summary

The validation tests indicate that MORD achieves its objectives, of relevancy, adaptability, and simplicity. As a secondary goal, an informal analysis of results of the tests confirm or reject informal hypotheses, based on general database theory, that have been proven for ordinary data in Oracle and which are likely to hold for multimedia data. Although the results from testing these hypotheses give an initial understanding of multimedia database performance, a more thorough evaluation is required to fully understand the performance of multimedia databases.

The validation tests were performed by altering the database configuration and/or the benchmark configuration and executing MORD in a controlled environment. Each test was run multiple times and results were collected. These results were statistically analysed to give an informal overview of MORD's capability and a broad idea of the performance of Oracle's multimedia database.

The informal analysis of the results suggests that MORD achieves each of its objectives and that it is easy to modify MORD so that it can evaluate a wide range of environments. As an aside, they suggest that better performance can be gained by using a larger block size, by making the buffer cache big enough to contain at least the largest expected dataset, and turning off caching and logging when possible.

The results demonstrate that MORD achieves each of its objectives and that it is easy to modify MORD so that it can evaluate a wide range of environments. As an aside, they suggest that better performance can be gained by using a larger block size, by making the buffer cache big enough to contain at least the largest expected dataset, and turning off caching and logging when possible.

Chapter 8 : Conclusion

Chapter 5 described the design of MORD, while chapter 6 detailed the design of tests to validate MORD. Chapter 7 presented the results with an informal analysis of running these validation tests. This chapter concludes the work done in this thesis, justifying it and explaining what it achieved. It summarises these contributions, and suggests areas for future work.

8.1 Motivation for Thesis

The rapid technological advancements in computing have led to an increase in the production and use of multimedia data, such as audio and video data. Multimedia data differs markedly from standard datatypes in both structure and required functionality, making it more difficult to manage. One method of managing such data is a Multimedia Database Management System (MMDBMS), which offers many advantages over alternative options. A MMDBMS can be defined as a DBMS that is able to store multimedia data internally as well as providing the necessary functionality to manipulate this data, which is not offered by standard DBMS, such as Relational DBMS (RDBMS). Object Relational Database Management Systems (ORDBMS) appear to be a suitable advanced DBMS, with the capability to handle multimedia data, while at the same time allowing for data to be queried using the simple SQL query language. Since multimedia databases, including ORDBMS, are still relatively new and there is limited knowledge on their working, further study of their performance is of particular interest.

A mechanism such as a benchmark is often used to measure the performance of any multimedia database. To measure the performance of a multimedia database, a benchmark should focus on what is common in a multimedia workload, such as large datasets and specialised queries, which differs extensively from a non-multimedia workload. A new benchmark that can be applied to different multimedia databases had to be designed.

8.2 Aim of Thesis

The goal of this thesis was to design a benchmark that can evaluate the performance of basic functionality found in multimedia databases, such as the multimedia application required by the Computer Science Department at Rhodes University. A Multimedia Object Relational Database (MORD) benchmark is designed for this purpose. MORD targets the Oracle ORDBMS specifically, but it is designed to be adaptable and can be used, with simple modifications, to benchmark any other Oracle multimedia database. For example, the test data used and queries tested by MORD can be altered.

8.3 Contributions of Thesis

MORD is a fully functional multimedia database benchmark that focuses on Basic functionality and includes database schemas, test data, and code to simulate queries and measure the response time and throughput of those queries. The schemas are written in SQL and create tables to store images, audio, and video data, which are all included in the test data. The test data is grouped into small, medium, and large categories for each of these multimedia datatypes. The queries that are evaluated by the code are insert, select, update, and delete. Other significant characteristics of MORD are its simplicity (making it easy to use and extend), scalability (making it possible to measure different sized databases).

Validation tests were performed to indicate that MORD functions correctly, as well as to demonstrate the manner in which it can be implemented. Verifying that MORD functions correctly requires testing that MORD meets all its objectives, of relevancy, adaptability, and simplicity. Through the validation of these objectives it is demonstrated that MORD's components, including schemas, test data, and queries can easily be adapted to suit various test environments, and that the presentation of the results can also be altered as necessary. More specifically it is demonstrated that MORD can:

- Connect to different databases
- Evaluate the performance of queries manipulating test data in a range of multimedia tables
- Evaluate different/additional multimedia tables by altering the schema
- Test various multimedia datatypes and formats

- Allow the user to add additional test data, including new types such as documents
- Vary the composition of the dataset
- Vary the amount of test data in terms of size or number of items
- Evaluate standard multimedia functionality, such as uploading data
- Evaluate the performance of specialised multimedia functionality, such as generating thumbnails
- Add additional functionality to be evaluated or alter the existing functionality
- Select the specific query type that is evaluated, such as insert
- Select the specific datatype that is evaluated, such as images
- Alter the precision and granularity of the timing mechanism
- Measure either response time or throughput

Together, these capabilities make MORD straightforward to use for a wide range of multimedia databases as well as easy to change or enhance when necessary.

As a secondary goal, the testing of MORD facilitated an initial understanding of Oracle's performance with respect to multimedia data. This was achieved by testing whether established database theory that holds for standard data also holds for multimedia data. Several informal hypotheses were formed, based on such database theory, and these were tested. An informal analysis of their results showed that, for Oracle:

- A larger block size improves performance
- Performance is generally related to the size of the data, especially for tasks such as uploading data, but not for tasks such as initialising an object
- Performance slows down if the size of the dataset is larger than the buffer cache
- Near linear scale-up is achievable and thresholds occur for certain increases in data
- If the caching and logging are turned off then the insert operation is faster
- Generating thumbnails takes longer than other tasks involved in insertion of images
- The time to read the header of some multimedia formats takes much longer than others, for example, it takes a long time to extract the properties of a PNG images

It must be noted that the testing of these informal hypotheses was only a secondary goal of the validation test and more in-depth testing and analysis would have to be performed using MORD to gain a better understanding of the performance of multimedia database.

8.4 Future Work

Future work could take several directions and includes:

- Applying MORD to help design a real system - MORD is to be used to assist in the configuration of the multimedia application for the Computer Science Department at Rhodes University. This will be achieved by evaluating the performance of a test environment that meets the requirements of the system. As expected, MORD can only be used for the initial tuning of the database as benchmarks are based on an estimation of the workload; fine tuning still has to be performed once the system is operational.

- Using MORD to compare different multimedia databases – Since MORD is portable, it could be adjusted as necessary and applied to two or more different types of multimedia databases. From the results for these applications of MORD it is possible to identify the strengths and weaknesses of each database. Recommendations can thus be made as to which multimedia database should ideally be used for which type of multimedia application.

- Extending MORD to test more advanced environments - MORD currently tests a single-user environment that is isolated from the network, and thus could be extended to test the performance of a multi-user, networked environment. To test such an environment requires careful consideration of many external factors that did not affect the MORD's test environment. For example, the performance of web based applications varies according to the speed of the network and this must be taken into consideration.

- Adding additional functionality to MORD - MORD could be extended to test additional functionality, such as signature generation or the cropping of images, and additional multimedia datatypes and format. This could easily be added in a similar manner to the other tests, just with the appropriate SQL command.

References

Benchmarks

[BD84] H. Boral and D.J..DeWitt. *A Methodology for Database System Performance Evaluation,* 1984.

[BDT83] D. Bitton, D.J. DeWitt and C. Turbyfill. *Benchmarking Database Systems: A Systematic Approach,* Proceedings in the Very Large Database Conference, 1983.

[BOT93] D. Bitton, C. Orji and C. Turbyfill. *AS^3AP – The ANSI SQL Standard Scalable and Portable Benchmark for Relational Database Systems,* In the Benchmark Handbook for Database and Transaction Processing Systems, Morgan Kaufmann, second edition, 1993.

[CAR97] M.J. Carey, D.J. DeWitt, J.F. Naughton, M. Asgarian, P. Brown, J.E. Gehrke and D.N. Shah. *The BUCKY Object-Relational Benchmark.* In Proceedings of the 1997 ACM-SIGMOD International Conference on the Management of Data, Tuscon, Arizona, May 1997

[CAT93] R.G.G Cattell. *The Engineering Database Benchmark,* In the Benchmark Handbook for Database and Transaction Processing Systems, J. Gray. Morgan Kaufmann, second edition, 1993.

[CDN93] M.J. Carey, D.J. DeWitt, J.F. Naughton. *The 007 Benchmark.* In Proceedings of the 1993 ACM-SIGMOD Conference on the Management of Data, Washington D.C.,May 1993.

[CHA95] A.B. Chaudhri. *An Annotated Bibliography of Benchmarks for Object Databases,* SIGMOD Vol. 24 No. 1 Pp 50 – 57, 1995.

[DEW93] D.J. DeWitt. *The Wisconsin Benchmark: Past, Present, and Future.* In the Benchmark Handbook for Database and Transaction Systems. Morgan Kaufmann, second edition, 1993.

[DH91] S. DeFazio and J. Hull. *Towards servicing textual database transactions on symmetric shared memory,* Proceedings in the fourth International Workshop on High Performance Transaction Processing Systems, Asilomar Conference Centre, September 1991.

[DIX93] K.M. Dixit. *Overview of* the *SPEC Benchmarks,* In the Benchmark Handbook for Database and Transaction Systems. Morgan Kaufmann, second edition, 1993.

[GGD95] A. Geppert, S. Gatziu, and K.R. Dittrich. *A Designer's Benchmark for Active Database Management Systems: 007 Meets the BEAST,* Proceedings in second Workshop on Rules in Databases, Athens, Greece, September 1995.

[GRA93] J. Gray. *The Benchmark Handbook for Database and Transaction Processing Systems.* Second edition, Morgan Kaufmann, San Fransico, 1993.

[LPM97] C. Lee, M. Potkonjak, W.H. Mangione-Smith. *MediaBench: A Tool for Evaluating and Synthesizing Multimedia and Communication Systems,* 1997.

[RIS00] N. Rishe, A. Vaschillo, D. Vasilevsky, A. Shaposhnikov, S. Chen. *A Benchmarking Technique for DBMS's with Advanced Data Models,* ADBIS-DASFAA Symposium, 2000 .

[RUN02] K. Runapongsa, J.M. Patel, H.V. Jagadish, and S. Al-Khalifa. *The Michigan Benchmark: Towards XML Query Performance Diagnostics*, February 2002.

[STO93] M. Stonebraker, J. Frew, K. Gardels, and J. Meredith. *The SEQUOIA 2000 Storage Benchmark.* Proceedings of the ACM SIGMOD International Conference on Management of Data, Washington DC, 1993.

[TRA02] Transaction Processing Performance Council benchmarks. May 2, 2002. [Online]. Available: http://www.tpc.org .

Database Tuning

[BMK99] P. Boncz, S. Manegold, and M. Kersten. *Database Architecture Optimized for the new Bottleneck: Memory Access,* Proceedings in the 25th VLDB Conference, Edinburgh, Scotland, 1999.

[KOR99] M. Kornacker. *High-Performance Extensible Indexing,* Proceedings in the 25th VLDB Conference, Edinburgh, Scotland, 1999.

[ORS96] B. Ozden, R. Rastogi, and A. Silberschantz. *Buffer Replacement Algorithms for Multimedia Storage Systems,* Proceedings in the ICMCS Conference, 1996.

[SB02] D. Shasha, and P. Bonnet. *Database Tuning: Principles, Experiments, and Troubleshooting Techniques.* Morgan Kaufmann, 2002.

Multimedia Databases

[CO70] E.F. Codd. *A Relational Model of Data for data of Large Shared Data Banks,* Proceedings in Communications in ACM, 1970.

[KA95] W. Klas, and K. Aberer. *Multimedia Applications and Implications on Database Architectures,* In advanced course on Multimedia Databases in Perspective, University of Twente, 1995.

[HA01] http://www.highwayafrica.org.za/archive/2001/.

[INF01] http://www-306.ibm.com/software/data/informix/ids/.

[MIT00] H. Mittag. *Multimedia and Multimedia Databases for Teaching Statistics,* In Proceedings in International Conference on Mathematical Education, July 2000.

[MMJ02] J. Melton, J. Michels, V. Josifovski, K. Kulkarni, P. Schwarz. *SQL/MED – A status report*, ACM SIGMOD, September 2002.

[SDB99] I. Soetebier, R. Dorner, and N. Braun. *Seamless Integration of Databases in VR for Constructing Virtual Environments,* Eurographics 99.

[SM99] M. Shapiro, and E. Miller. *Managing Databases with Binary Large Objects,* Proceedings in the 16th IEEE Mas Storage Symposium, March 1999, San Diego.

[STO96] M. Stonebraker. *Object-Relational Database Systems: The Next Wave*. Morgan Kaufmann, San Francisco, 1996.

[SUB98] V. Subrahmanian. *Principles of Multimedia Database Systems,* Morgan Kaufmann Publishers, Inc., San Fransico, California, 1998.

Oracle

[BGB98] L.A. Barroso, K. Gharachorloo, and E. Bugnion. *Memory System Characterization of Commercial Workloads,* Proceedings in the 25th Symposium on Computer Architecture, June 1998.

[BRF01] R. Baylis, K. Rich, and J. Fee. *Oracle9i Database Administrators Guide,* Oracle Corporation Documentation, June 2001.

[CG01] M. Cyran, and C.D. Green, *Oracle9i Database Performance Guide and Reference,* Oracle Corporation Documentation, June 2001.

[CHI01] M. Chittister, *Oracle9i interMedia Java Classes User's Guide and Reference,* Oracle Corporation Documentation, June 2001.

[DZ02] B. Dageville, and M. Zait. *SQL Memory Management in Oracle9i,* Proceedings in the 28th VLDB Conference, Hong Kong, China, 2002.

[KL99] D.C. Kreines, and B. Laskey. *Oracle Database Administrator, The Essential Reference*, O'Reilly & Associates, Inc.,1999.

[LO01] J. Loaiza. *Optimal Storage Configuration Made Easy*, Oracle Corporation White Paper, June 2001.

[MCG01] L. McGee Luscher *Oracle 9i Database Concepts,* Oracle Corporation Documentation, June 2001.

[OIM02] *Oracle interMedia Media Storage & Retrieval Performance*, September 2002, Oracle Corporation White Paper, http://otn.oracle.com/products/intermedia/htdocs/perf_summary.html.

[OTN02] http://otn.oracle.com/deploy/performance/content.html.

Performance Evaluations

[HPS00] I. Horrocks, P. Patel-Schneider, and R. Sebastiani. *An Analysis of Empirical Testing for Modal Decision Procedures,* In IGPL, Vol. 8 No. 3, pp. 293-323, 2000.

[JM95] B. Jacob, and T. Mudge. *Notes on Calculating Computer Performance,* University of Michigan Tech report, March 1995.

[MC99] R. Musick, and T. Critchlow. *Practical Lessons in Supporting Large-Scale Computational Science,* December 99, SIGMOD.

Multimedia Data

[ADO02]http://www.adobe.com/support/techguides/webpublishing/audio/page3.html.

[FRA01] http://www.iis.fraunhofer.de/amm/techinf/layer3/.

[JPG02] http://www.faqs.org/faqs/jpeg-faq/.

[MCG02] J.F. McGowan, Ph.D. *AVI Overview,* 2002, http://www.jmcgowan.com/.

[PNG02] http://www.libpng.org/pub/png/.

[STA98] http://acomp.stanford.edu/acpubs/Docs/graphic_file_formats/.

[SON00] Y. Song, Arizona State University, Video Streaming Applications: QuickTime, Real, and Windows Media, 2000.

[W3C02] http://www.w3c.org.

Appendix A: Outline of MORD's Code

A.1 Image Response Time Code

```
public class ImageExperiments {
      ImageExperiments(String dbname, String dbuser, String dbpass)   {  }
      ImageExperiments() { }
      public OracleConnection connect() throws Exception  { }
      public  void  insertStm(OracleConnection  con,  String  imgNm,  String
      imgDir, String imgType, int num, BufferedWriter f)  { }
      public  void  selectStm(OracleConnection  con,  String  imgSize,  String
      imgType, int num, BufferedWriter f) {}
      public  void  updateStm(OracleConnection  con,  String  imgSize,  String
      imgType, int num, BufferedWriter f)  { }
      public  void  deleteStm(OracleConnection  con,  String  imgSize,  String
      imgType, int num, BufferedWriter f )  { }
      public void callInsert(BufferedWriter D, OracleConnection con)  { }
      public void callSelect(BufferedWriter D, OracleConnection con)  { }
      public void callUpdate(BufferedWriter D, OracleConnection con)  { }
      public void callDelete(BufferedWriter D, OracleConnection con)  { }
      public void control (String cmd)  { }
}
```

A.2 Image Throughput Code

```
public class ImageTP {
      ImageTP(String  dbname,  String  dbuser,  String  dbpass)     {    }
      ImageTP() { }
      public OracleConnection connect() throws Exception  { }
      public  void  insertStm(OracleConnection  con,  String  imgNm,  String
      imgDir, String imgType, int timing, BufferedWriter f)  { }
      public  void  selectStm(OracleConnection  con,  String  imgSize,  String
      imgType, int timing, BufferedWriter f) {}
      public  void  updateStm(OracleConnection  con,  String  imgSize,  String
      imgType, int timing, BufferedWriter f)  { }
      public  void  deleteStm(OracleConnection  con,  String  imgSize,  String
      imgType, int timing, BufferedWriter f )  { }
      public void callInsert(BufferedWriter D, OracleConnection con)  { }
      public void callSelect(BufferedWriter D, OracleConnection con)  { }
      public void callUpdate(BufferedWriter D, OracleConnection con)  { }
      public void callDelete(BufferedWriter D, OracleConnection con)  { }
      public void control (String cmd)  { }
}
```

A.3 Audio Response Time Code

```
public class AudioExperiments {
      AudioExperiments(String dbname, String dbuser, String dbpass)   {  }
      AudioExperiments() { }
      public OracleConnection connect() throws Exception  { }
      public  void  insertStm(OracleConnection  con,  String  audNm,  String
      audDir, String audType, int num, BufferedWriter f)  { }
      public  void  selectStm(OracleConnection  con,  String  audNm,  String
      audDir, String audType, int num, BufferedWriter f) {}
      public  void  updateStm(OracleConnection  con,  String  audNm,  String
      audDir, String audType, int num, BufferedWriter f)  { }
      public  void  deleteStm(OracleConnection  con,  String  audNm,  String
      audDir, String audType, int num, BufferedWriter f )  { }
      public void callInsert(BufferedWriter D, OracleConnection con)  { }
      public void callSelect(BufferedWriter D, OracleConnection con)  { }
      public void callUpdate(BufferedWriter D, OracleConnection con)  { }
      public void callDelete(BufferedWriter D, OracleConnection con)  { }
      public void control (String cmd)  { }
}
```

A.4 Audio Throughput Code

```
public class AudioTP {
      AudioExperiments(String dbname, String dbuser, String dbpass)   {  }
      AudioExperiments() { }
      public OracleConnection connect() throws Exception  { }
      public  void  insertStm(OracleConnection  con,  String  audNm,  String
      audDir, String audType, int timing, BufferedWriter f)  { }
      public  void  selectStm(OracleConnection  con,  String  audNm,  String
      audDir, String audType, int timing, BufferedWriter f) {}
      public  void  updateStm(OracleConnection  con,  String  audNm,  String
      audDir, String audType, int timing, BufferedWriter f)  { }
      public  void  deleteStm(OracleConnection  con,  String  audNm,  String
      audDir, String audType, int timing, BufferedWriter f )  { }
      public void callInsert(BufferedWriter D, OracleConnection con)  { }
      public void callSelect(BufferedWriter D, OracleConnection con)  { }
      public void callUpdate(BufferedWriter D, OracleConnection con)  { }
      public void callDelete(BufferedWriter D, OracleConnection con)  { }
      public void control (String cmd)  { }
}
```

A.5 Video Response Time Code

```
public class VideoExperiments {
	VideoExperiments(String dbname, String dbuser, String dbpass)   {  }
	VideoExperiments() { }
	public OracleConnection connect() throws Exception  { }
	public void insertStm(OracleConnection con, String vidNm, String vidDir, String vidType, int num, BufferedWriter f)  { }
	public void selectStm(OracleConnection con, String vidNm, String vidDir, String vidType, int num, BufferedWriter f) {}
	public void updateStm(OracleConnection con, String vidNm, String vidDir, String vidType, int num, BufferedWriter f)  { }
	public void deleteStm(OracleConnection con, String vidNm, String vidDir, String vidType, int num, BufferedWriter f )  { }
	public void callInsert(BufferedWriter D, OracleConnection con)  { }
	public void callSelect(BufferedWriter D, OracleConnection con)  { }
	public void callUpdate(BufferedWriter D, OracleConnection con)  { }
	public void callDelete(BufferedWriter D, OracleConnection con)  { }
	public void control (String cmd)  { }
}
```

A.6 Video Throughput Code

```
public class VideoTP {
	VideoTP(String dbname, String dbuser, String dbpass) { }
	VideoTP() { }
	public OracleConnection connect() throws Exception  { }
	public void insertStm(OracleConnection con, String vidNm, String vidDir, String vidType, int timing, BufferedWriter f)  { }
	public void selectStm(OracleConnection con, String vidNm, String vidDir, String vidType, int timing, BufferedWriter f) {}
	public void updateStm(OracleConnection con, String vidNm, String vidDir, String vidType, int timing, BufferedWriter f)  { }
	public void deleteStm(OracleConnection con, String vidNm, String vidDir, String vidType, int timing, BufferedWriter f )  { }
	public void callInsert(BufferedWriter D, OracleConnection con)  { }
	public void callSelect(BufferedWriter D, OracleConnection con)  { }
	public void callUpdate(BufferedWriter D, OracleConnection con)  { }
	public void callDelete(BufferedWriter D, OracleConnection con)  { }
	public void control (String cmd)  { }
}
```

A.7 Main Classes

```
public class MultiUser
{
      public static void main(String args[]){ }
}


class Test implements Runnable
{
      public Test(String str, int num1 ){ }
      public void run() { }
}
```

A.8 Schema Code in Generic.Sql

```
----------------------------------------------------
-- Drop any tables already created
----------------------------------------------------
DROP TABLE VIDEO;
DROP TABLE IMAGE;
DROP TABLE AUDIO;
DROP TABLE DOCUMENT;

----------------------------------------------------
-- Drop any sequences already created
----------------------------------------------------
DROP SEQUENCE VIDEO_sequence;
DROP SEQUENCE IMAGE_sequence;
DROP SEQUENCE AUDIO_sequence;
DROP SEQUENCE DOCUMENT_sequence;

----------------------------------------------------
-- Create the sequence for the video table
----------------------------------------------------
CREATE SEQUENCE VIDEO_sequence
      INCREMENT BY 1
      START WITH 1
      NOMAXVALUE
      NOCYCLE
      CACHE 10;

----------------------------------------------------
-- Create the video table
----------------------------------------------------
CREATE TABLE VIDEO(     VidID            NUMBER  PRIMARY KEY,
                  VideoData   ORDSYS.ORDVideo,
                  VideoName   VARCHAR2(200));

----------------------------------------------------
-- Create the sequence for the image table
----------------------------------------------------
CREATE SEQUENCE IMAGE_sequence
      INCREMENT BY 1
      START WITH 1
      NOMAXVALUE
      NOCYCLE
      CACHE 10;

----------------------------------------------------
-- Create the image table
----------------------------------------------------
CREATE TABLE IMAGE(     ImgID            NUMBER  PRIMARY KEY,
                  ImageData   ORDSYS.ORDImage,
                  ImageName   VARCHAR2(200)
                  );

----------------------------------------------------
-- Create the sequence for the audio table
----------------------------------------------------
CREATE SEQUENCE AUDIO_sequence
      INCREMENT BY 1
      START WITH 1
      NOMAXVALUE
      NOCYCLE
      CACHE 10;

----------------------------------------------------
-- Create the audio table
----------------------------------------------------
CREATE TABLE AUDIO(     AudioID     NUMBER  PRIMARY KEY,
                  AudioData   ORDSYS.ORDAUDIO,
                  AudioTitle  VARCHAR2(200));
```

A.9 Constructors and Code to Connect to Database for Response Time and Throughput Experiments

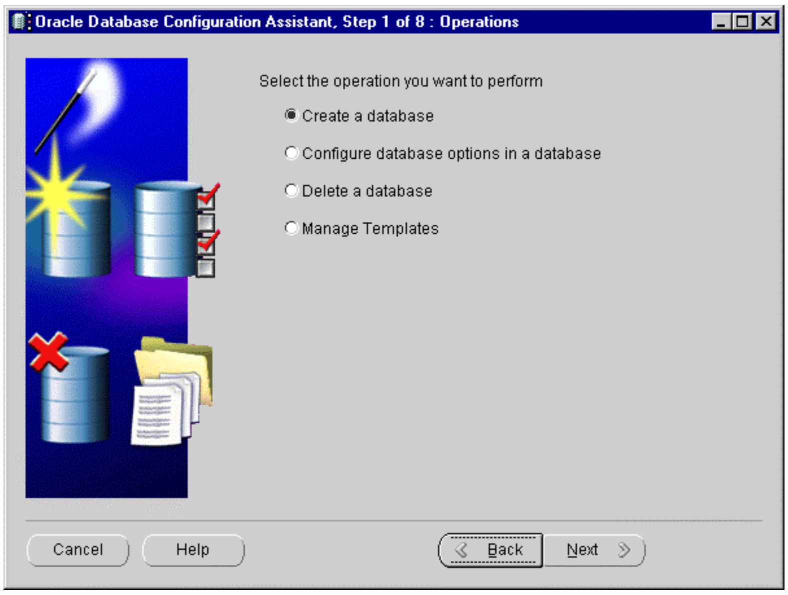
```
String dname, duser,dpass;
ImageExperiments(String dbname, String dbuser,String dbpass)
{
      dname=dbname;
      duser=dbuser;
      dpass=dbpass;
}

ImageExperiments()
{
      dname="large";
      duser="mmedia";
      dpass="mmedia";
}

public OracleConnection connect() throws Exception
{
      String connectString;
      Class.forName ("oracle.jdbc.driver.OracleDriver");
      System.out.println(dname);
      connectString = "jdbc:oracle:thin:@nemesis.ict.ru.ac.za:1521:" +
dname;
      OracleConnection con = (OracleConnection)
            DriverManager.getConnection(connectString,duser,dpass);
      con.setAutoCommit(false);
      return con;
  }
```

Appendix B: Installation Instructions

The instructions given in this section start from after Oracle has been installed and the user has logged onto Oracle. They are the steps that need to be taken to setup the environment for MORD to be executed. It is also presumed that the code and the test data have been copied across in the directories as they were. The main directory is called Benchmark where MORD's code is and there are 4 subdirectories called Images, Audio, Video, and Documents. Each of these subdirectories has 3 further subdirectories called small, medium and large where the test data resides.

B.1 Step 1: Create Database

The easiest way to create the databases is by using an Oracle wizard. This is a 8 step process which starts with the screen shown in Figure B-1, where the option to create a database must be selected.

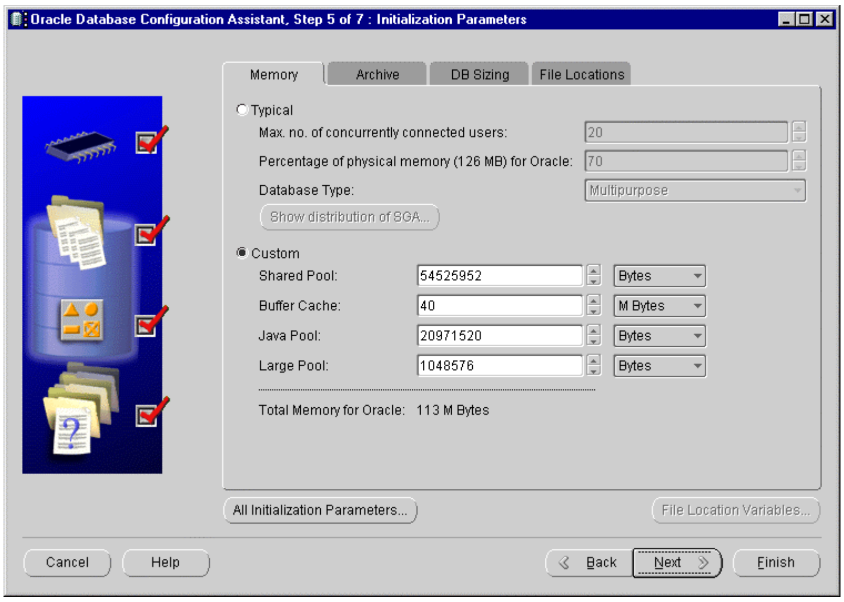

Oracle Database Configuration Assistant, Step 1 of 8 : Operations

Select the operation you want to perform

[✓ Create a database]
Configure database options in a database
Delete a database
Manage Templates

B-1 Screenshot of first screen for Oracle database wizard

The steps are straight forward with most of them just being left as the default setting and clicking on the next button. Step 5 is important as this step allows the user to change the sizes of the caches, such as the buffer cache. The screenshot in B-2 shows the screen from which this is carried out.

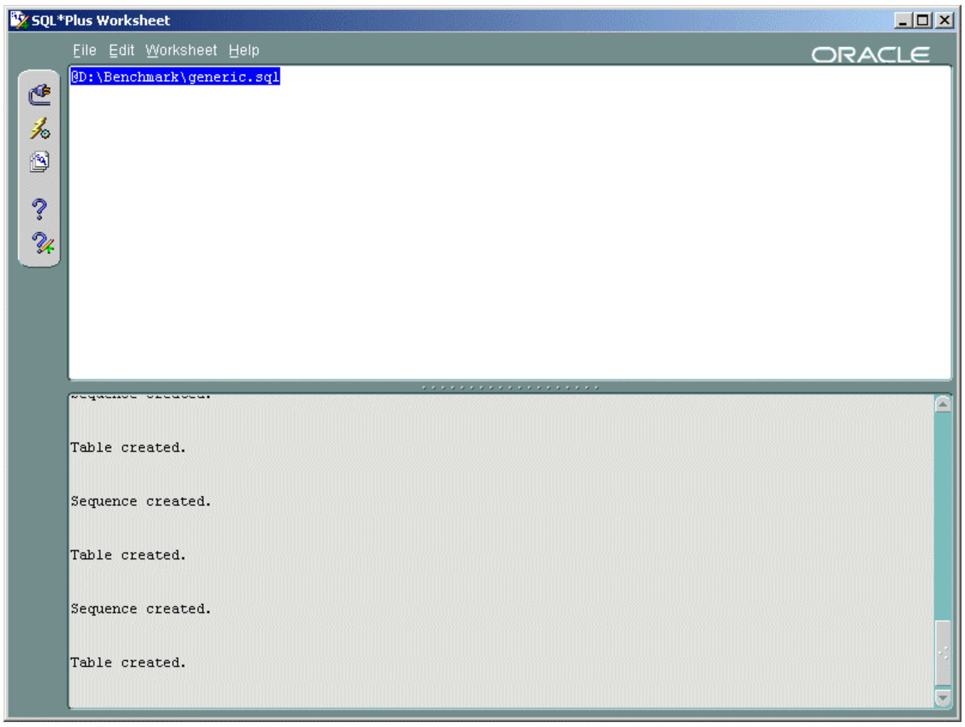

Oracle Database Configuration Assistant, Step 5 of 7 : Initialization Parameters
Memory
Archive
DB Sizing
File Locations
All Initialization Parameters...
File Location Variables...
Cancel
Help
Back
Next
Finish

Typical
Max. no. of concurrently connected users: 20
Percentage of physical memory (126 MB) for Oracle: 70
Database Type: Multipurpose
Show distribution of SGA...
Custom
Shared Pool: 54525952 Bytes
Buffer Cache: 40 M Bytes
Java Pool: 20971520 Bytes
Large Pool: 1048576 Bytes
Total Memory for Oracle: 113 M Bytes

B-2 Screenshot of step 5 for database creation

B.2 Step 2: Generate Tables

The schemas must be generated by executing the schema script `generic.sql`. It can be run using the SQL*Plus Worksheet as shown in Figure B-3.



```
@D:\Benchmark\generic.sql
```

```
Table created.

Sequence created.

Table created.

Sequence created.

Table created.
```

*B-3 Screenshot of SQL*Plus worksheet to generate database schema*

B.3 Step 3: Setup Test Data

To setup the test data correctly Oracle has to be configured so that it can upload data from a directory. This is achieved by creating the directory in Oracle and granting the database access to it. Code must be executed to make this directory known to the database, such as the sample code given in Figure B-4.

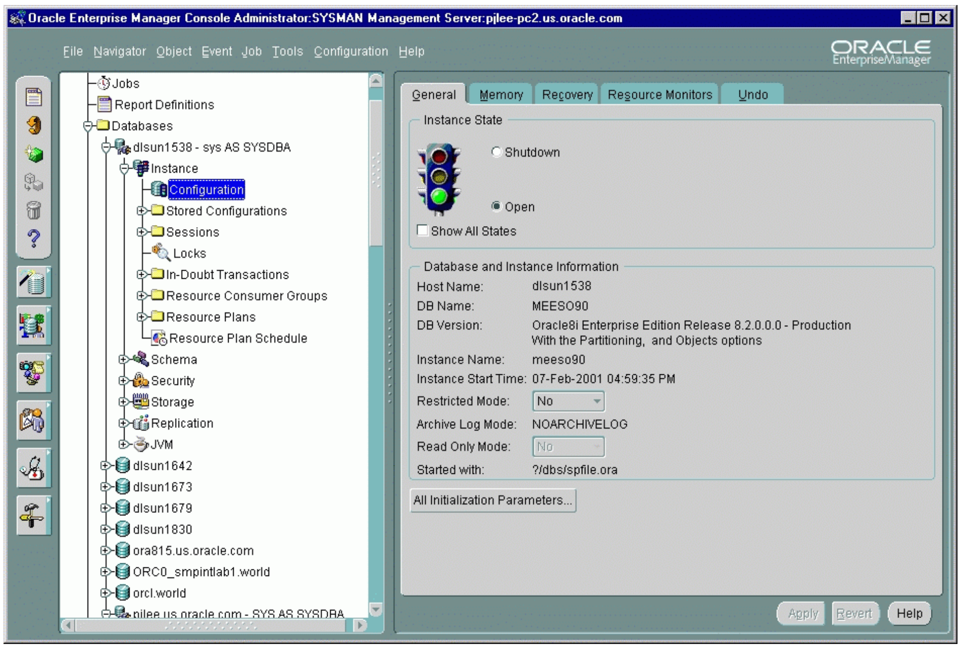
```
create or replace directory ORDDOCDIR as ‘DIRECTORY LOCTATION’;
grant read on directory ORDDOCDIR to public with grant option;
```

B-4 Code to make Oracle aware of the directory

To check that this has been executed correctly and that Oracle has access to this directory, the `Sys.Dir$` table can be queried.

B.4 Step 4: Configure Database

The database must be configured as specified in Table 5-2, and can be achieved from the screen shown in Figure B-5.



Instance State
Shutdown
[✓ Open]
Show All States

Database and Instance Information

Host Name:	dlsun1538
DB Name:	MEES090
DB Version:	Oracle8i Enterprise Edition Release 8.2.0.0.0 - Production With the Partitioning, and Objects options
Instance Name:	meeso90
Instance Start Time:	07-Feb-2001 04:59:35 PM
Restricted Mode:	No
Archive Log Mode:	NOARCHIVELOG
Read Only Mode:	No
Started with:	?/dbs/spfile.ora

B-5 Screenshot of Oracle screen for configuring the database

The button at the bottom of this window must be selected to alter the initialisation parameters. Here the OPEN_CURSORS must be set to 3000 and the SGA_MAX_SIZE must be made large enough for the experiments, depending on what size caches are being tested. The memory tab can then be select to configure the various caches, such as the Java pool.

B.5 Step 5: Compile and Run MORD

To compile and run MORD’s code, the following commands must be executed from the code’s directory:

```
javac MultiUser.java
java MultiUser
ALL (if all operations/datatypes)
```

Appendix C: Tables of Results for Hypotheses

C.1 MORD's Measurements are Repeatable

Test 1:Table showing the average variation for Images

		Insert	Select	Update	Delete
Small	Bmp	1.925%	5.460%	5.855%	3.097%
	Gif	0.690%	3.632%	5.029%	1.964%
	Jpg	1.045%	4.638%	1.494%	2.913%
	Png	0.340%	0.280%	0.316%	2.717%
	Tif	1.625%	4.550%	1.352%	3.107%
Medium	Bmp	1.439%	3.810%	0.512%	1.760%
	Gif	0.579%	4.729%	0.481%	2.742%
	Jpg	1.026%	4.487%	0.313%	1.466%
	Png	0.282%	4.340%	0.072%	3.683%
	Tif	1.091%	4.729%	0.927%	2.310%
Large	Bmp	1.767%	4.521%	1.250%	2.417%
	Gif	3.654%	2.796%	1.026%	0.707%
	Jpg	5.621%	0.280%	0.794%	1.045%
	Png	2.041%	3.636%	0.144%	1.987%
	Tif	3.479%	2.563%	1.289%	1.512%
	Average	1.77%	3.63%	1.39%	2.23%

Category Averages	
Insert	1.13%
Select	3.71%
Update	2.81%
Delete	2.76%
Insert	0.88%
Select	4.42%
Update	0.46%
Delete	2.39%
Insert	3.31%
Select	2.76%
Update	0.90%
Delete	1.53%

Test 1: Table showing the variation for Audio Data

		Insert	Select	Update	Delete
Small	wav	0.1572%	4.7640%	0.7456%	3.8350%
	mp3	1.1487%	2.4771%	1.4718%	2.4787%
	au	1.6185%	2.9800%	3.1272%	5.7058%
Medium	wav	0.3923%	1.9961%	0.6209%	5.5432%
	mp3	0.5266%	3.0401%	0.0697%	3.4842%
	au	1.0737%	3.2643%	2.5321%	3.6948%
Large	wav	1.3861%	0.6681%	2.5965%	1.1895%
	mp3	0.6722%	1.2761%	0.0255%	0.2881%
	au	0.1885%	2.1750%	7.2375%	1.2217%
	Average	0.80%	2.52%	2.05%	3.05%

Category Averages	
Insert	0.97%
Select	3.41%
Update	1.78%
Delete	4.01%
Insert	0.66%
Select	2.77%
Update	1.07%
Delete	4.24%
Insert	0.75%
Select	1.37%
Update	3.29%
Delete	0.90%

Test1: Table showing the variation for Video Data

		Insert	Select	Update	Delete
Small	avi	0.1404%	2.8602%	0.5034%	3.0332%
	mov	0.1520%	1.1156%	0.4819%	4.4873%
	rm	0.0880%	1.1689%	0.2773%	5.7724%
Medium	avi	0.7242%	3.3053%	1.1103%	1.1475%
	mov	0.1759%	2.2294%	0.9881%	3.1149%
	ram	0.6202%	0.9615%	0.6831%	1.5738%
Large	avi	0.1478%	0.2146%	0.2293%	0.5995%
	mov	0.4838%	0.2318%	1.0098%	5.1525%
	ram	0.4470%	0.4709%	0.8925%	2.0164%
	Average	0.33%	1.40%	0.69%	2.99%

Category Averages	
Insert	0.13%
Select	1.71%
Update	0.42%
Delete	4.43%
Insert	0.51%
Select	2.17%
Update	0.93%
Delete	1.95%
Insert	0.36%
Select	0.31%
Update	0.71%
Delete	2.59%

C.2 Specified Queries can be Tested by MORD

Test 3:Table showing the average time taken for images with no caching

		Insert
Small	Bmp	119.6600
	Gif	122.9300
	Jpg	117.2000
	Png	930.9300
	Tif	135.6800
Medium	Bmp	808.2900
	Gif	1,039.7000
	Jpg	746.9900
	Png	11,468.4500
	Tif	728.1700
Large	Bmp	11,576.4000
	Gif	9,889.8400
	Jpg	6,853.1200
	Png	9,471.6300
	Tif	6,898.3600
	Average	4,060.490

Category Averages	
Insert	285.280
Insert	2,958.320
Insert	8,937.870

Test 3: Table showing the average time taken for audio with no caching

		Insert
Small	wav	1,547.3600
	mp3	871.1600
	au	696.9600
Medium	wav	8,435.6400
	mp3	4,696.8400
	au	3,598.1600
Large	wav	102,980.6400
	mp3	74,122.4400
	au	44,487.5000
	Average	26,826.300

Category Averages	
Insert	1,038.493
Insert	5,576.880
Insert	73,863.527

Test 3: Table showing the average time taken for video with no caching

		Insert
Small	avi	7,652.12
	mov	6,927.48
	rm	17,101.92
Medium	avi	36,852.48
	mov	36,383.76
	ram	45,050.00
Large	avi	532,093.08
	mov	168,730.04
	ram	180,646.24
	Average	114,604

Category Averages	
Insert	10,561
Insert	39,429
Insert	293,823

Test 3: Table showing the average time taken for images with caching

		Insert
Small	Bmp	147.6500
	Gif	311.2400
	Jpg	157.6700
	Png	928.9300
	Tif	109.6900
Medium	Bmp	615.0100
	Gif	378.1200
	Jpg	377.1600
	Png	4,752.1800
	Tif	4,752.1800
Large	Bmp	6,771.7100
	Gif	4,902.5800
	Jpg	4,394.7100
	Png	33,925.0000
	Tif	6,730.7700
	Average	4,616.973

Category Averages	
Insert	331.036
Insert	2,174.930
Insert	11,344.954

Test 3: Table showing the average time taken for audio with caching

		Insert
Small	wav	2,348.76
	mp3	973.16
	au	969.32
Medium	wav	16,762.00
	mp3	5,140.60
	au	4,045.72
Large	wav	107,138.08
	mp3	99,840.56
	au	61,471.10
	Average	33,188

Category Averages	
Insert	1,430
Insert	8,649
Insert	89,483

Test 3: Table showing the average time taken for video with caching

		Insert
Small	avi	10,730.0000
	mov	13,774.9600
	rm	28,752.3600
Medium	avi	57,127.4800
	mov	44,940.0400
	ram	51,518.0800
Large	avi	637,986.9200
	mov	186,689.3200
	ram	199,440.0400
	Average	136,773.244

Category Averages	
Insert	17,752.440
Insert	51,195.200
Insert	341,372.093

C.3 MORD’s Dataset can be Altered

Test 4: Table showing the average time taken for Dataset 1/Buffer Cache 320

		Select
Small	Bmp	7.76
	Gif	7.82
	Jpg	7.96
	Png	7.99
	Tif	7.50
Medium	Bmp	7.76
	Gif	8.08
	Jpg	7.97
	Png	8.07
	Tif	8.01
Large	Bmp	8.61
	Gif	8.44
	Jpg	8.58
	Png	8.47
	Tif	9.07

Category Averages	
Select	7.81
Select	7.98
Select	8.63
Average	8.14

Test 4: Table showing the average time taken Dataset 2/ Buffer Cache 3

Small	Bmp	7.58
	Gif	7.56
	Jpg	7.66
	Png	7.33
	Tif	7.49
Medium	Bmp	7.66
	Gif	7.65
	Jpg	7.51
	Png	7.65
	Tif	7.64
Large	Bmp	8.04
	Gif	7.56
	Jpg	7.35
	Png	7.49
	Tif	7.56

Select	7.52
Select	7.62
Select	7.60
Average	7.58

Test 4: Table showing the average time taken for Dataset 3/Buffer Cache 320

		Select
Small	Bmp	7.63
	Gif	7.36
	Jpg	7.18
	Png	7.17
	Tif	7.18
Medium	Bmp	7.51
	Gif	7.02
	Jpg	6.86
	Png	8.11
	Tif	7.36
Large	Bmp	7.35
	Gif	7.35
	Jpg	7.40
	Png	7.46
	Tif	7.50

Category Averages	
Select	7.30
Select	7.37
Select	7.41
Average	7.36

Test 4: Table showing the average time taken Dataset 1/Buffer Cache 100

		Select
Small	Bmp	8.09
	Gif	7.74
	Jpg	8.14
	Png	7.79
	Tif	8.04
Medium	Bmp	8.28
	Gif	8.44
	Jpg	8.38
	Png	9.06
	Tif	8.28
Large	Bmp	8.75
	Gif	8.79
	Jpg	8.76
	Png	9.08
	Tif	8.92

Category Averages	
Select	7.96
Select	8.49
Select	8.86
Average	8.44

Test 4: Table showing the average time taken for Dataset 2/Buffer Cache 100

		Select
Small	Bmp	7.96
	Gif	7.50
	Jpg	7.82
	Png	7.35
	Tif	7.81
Medium	Bmp	8.28
	Gif	7.51
	Jpg	7.50
	Png	7.49
	Tif	7.82
Large	Bmp	8.43
	Gif	7.65
	Jpg	8.12
	Png	7.97
	Tif	7.98

Category Averages	
Select	7.69
Select	7.72
Select	8.03
Average	7.81

Test 4: Table showing the average time taken for Dataset 2/Buffer Cache 100

		Select
Small	Bmp	7.32
	Gif	7.51
	Jpg	7.14
	Png	7.18
	Tif	7.28
Medium	Bmp	7.81
	Gif	7.33
	Jpg	7.83
	Png	7.02
	Tif	7.19
Large	Bmp	7.68
	Gif	7.34
	Jpg	7.35
	Png	7.65
	Tif	7.33

Category Averages	
Select	7.29
Select	7.44
Select	7.47
Average	7.40

Test 4: Table showing the average time taken for Dataset 1/Buffer Cache 16

		Select
Small	Bmp	7.97
	Gif	8.03
	Jpg	7.97
	Png	7.98
	Tif	7.96
Medium	Bmp	9.53
	Gif	9.16
	Jpg	8.87
	Png	8.76
	Tif	8.44
Large	Bmp	9.84
	Gif	9.05
	Jpg	9.53
	Png	9.07
	Tif	9.61

Category Averages	
Select	7.98
Select	8.95
Select	9.42
Average	8.78

Test 4: Table showing the average time taken for Dataset 2/Buffer Cache 16

		Select
Small	Bmp	7.65
	Gif	7.56
	Jpg	7.64
	Png	7.54
	Tif	7.62
Medium	Bmp	8.27
	Gif	8.18
	Jpg	8.18
	Png	8.27
	Tif	8.24
Large	Bmp	8.44
	Gif	8.35
	Jpg	8.42
	Png	8.28
	Tif	8.43

Category Averages	
Select	7.60
Select	8.23
Select	8.38
Average	8.07

Test 4:Table showing the average time taken for Dataset 3/Buffer Cache 16

		Select
Small	Bmp	7.51
	Gif	7.34
	Jpg	7.80
	Png	7.51
	Tif	7.22
Medium	Bmp	8.14
	Gif	7.13
	Jpg	7.51
	Png	7.65
	Tif	7.19
Large	Bmp	8.89
	Gif	7.57
	Jpg	7.86
	Png	7.49
	Tif	7.71

Category Averages	
Select	7.48
Select	7.52
Select	7.90
Average	7.63

C.4 MORD can Connect to a Different Database

Test 6: Table showing the average time taken for Image when block size = 4K

	Insert	Select	Update	Delete
Bmp	135.61	11.24	83.66	23.29
Gif	152.29	9.07	86.00	23.75
Jpg	157.81	9.54	115.47	23.43
Png	989.26	9.22	909.99	23.75
Tif	167.03	9.37	106.47	23.57
Bmp	885.94	9.39	526.89	25.32
Gif	612.83	9.23	479.68	26.10
Jpg	561.55	9.07	567.19	29.21
Png	5682.19	9.06	4929.07	28.59
Tif	547.65	8.98	572.96	28.50
Bmp	6545.00	9.18	4485.48	61.10
Gif	7213.91	9.22	3991.40	65.32
Jpg	6659.37	9.38	4458.30	64.69
Png	35109.86	9.22	32044.67	69.23
Tif	9887.03	9.22	4387.66	60.62
Average	5020.49	9.36	3849.66	38.43

Category Averages	
Insert	320.40
Select	9.69
Update	260.32
Delete	23.56

Insert	1,658.03
Select	9.15
Update	1,415.16
Delete	27.54

Insert	13,083.03
Select	9.24
Update	9,873.50
Delete	64.19

Test 6: Table showing the average time taken for audio when block size=4k

	Insert	Select	Update	Delete
wav	2276.24	354.40	539.32	35.64
mp3	1511.28	314.08	408.08	33.12
au	996.56	216.76	111.52	34.92
wav	11935.64	2014.36	139.32	64.96
mp3	5708.16	1717.48	2305.08	61.92
au	4067.92	1351.20	88.12	65.60
wav	88320.44	38879.48	181.24	498.16
mp3	99365.04	36752.40	25151.88	317.52
au	65050.75	24039.85	99.55	329.05
Average	31025.78	11737.78	3224.90	160.10

Category Averages	
Insert	1594.69
Select	295.08
Update	352.97
Delete	34.56
Insert	7237.24
Select	1694.35
Update	844.17
Delete	64.16
Insert	84245.41
Select	33223.91
Update	8477.56
Delete	381.58

Test 6: Table showing the average time taken for video when block size=4k

	Insert	Select	Update	Delete
avi	12975.64	1472.04	3786.84	47.52
mov	13814.96	1601.36	3895.64	48.16
rm	19936.36	1408.04	6818.72	54.44
avi	91845.12	15006.88	23664.36	157.16
mov	44940.04	21377.60	24970.00	159.16
ram	51518.08	20388.40	52468.12	408.84
avi	454827.52	148735.40	388385.68	2197.04
mov	174689.32	60600.60	77368.76	418.48
ram	175440.04	59679.24	75560.60	804.72
Average	115554.12	36696.62	72990.97	477.28

Category Averages	
Insert	15575.65
Select	1493.81
Update	4833.73
Delete	50.04
Insert	62767.75
Select	18924.29
Update	33700.83
Delete	241.72
Insert	268318.96
Select	89671.75
Update	180438.35
Delete	1140.08

Test 6: Table showing the average time taken for Images when block size = 8k

		Insert	Select	Update	Delete
Small	Bmp	135.00	8.83	63.44	22.60
	Gif	121.43	9.04	57.83	22.90
	Jpg	157.67	9.07	97.65	22.64
	Png	930.93	8.74	860.00	22.23
	Tif	109.69	8.90	83.92	22.66
Medium	Bmp	615.01	8.75	493.75	24.91
	Gif	399.12	8.59	460.93	24.52
	Jpg	395.16	8.75	565.61	24.22
	Png	4752.18	8.59	4727.04	23.41
	Tif	363.45	8.06	520.62	27.27
Large	Bmp	6360.85	8.61	4204.23	41.25
	Gif	4913.58	9.06	3631.56	47.50
	Jpg	5394.71	9.22	4390.62	43.44
	Png	33925.00	8.78	29800.14	54.36
	Tif	6730.77	9.07	4112.99	45.63
	Average	4353.64	8.80	3604.69	31.30

Category Averages	
Insert	290.94
Select	8.92
Update	232.57
Delete	22.61

Insert	1,304.98
Select	8.55
Update	1,353.59
Delete	24.87

Insert	11,464.98
Select	8.95
Update	9,227.91
Delete	46.44

Test 6: Table showing the average time taken for Audio when block size=8k

		Insert	Select	Update	Delete
Small	wav	2,228.7600	350.6400	538.2000	31.2800
	mp3	913.1600	214.4400	309.4800	31.1200
	au	969.3200	201.8800	84.4000	27.5200
Medium	wav	9,962.0000	1,639.9600	131.8800	53.1600
	mp3	4,900.6000	1,539.4000	2,294.4000	48.8400
	au	4,045.7200	1,178.8000	85.6000	41.2400
Large	wav	82,418.0800	38,300.6400	175.7200	299.3600
	mp3	99,040.5600	29,873.1600	24,625.6400	211.5200
	au	51,471.1000	15,274.2500	84.3500	283.5500
	Average	28,438.811	9,841.463	3,147.741	114.177

Category Averages	
Insert	1,370.413
Select	255.653
Update	310.693
Delete	29.973
Insert	6,302.773
Select	1,452.720
Update	837.293
Delete	47.747
Insert	77,643.247
Select	27,816.017
Update	8,295.237
Delete	264.810

Test 6: Table showing the average time taken for Video when block size=8k

		Insert	Select	Update	Delete
Small	avi	10730.00	1262.52	3761.28	33.76
	mov	9090.64	1316.16	3818.72	33.40
	rm	14317.48	1199.36	5308.16	39.36
Medium	avi	62727.48	10992.56	22446.32	108.16
	mov	43088.72	17626.00	24613.80	103.76
	ram	36430.00	16332.52	39077.48	326.28
Large	avi	445986.92	140578.12	272705.04	1575.64
	mov	158714.36	56040.64	70116.28	316.92
	ram	153608.76	58423.76	68347.48	551.92
	Average	103854.93	33752.40	56688.28	343.24

Category Averages	
Insert	11,379.37
Select	1,259.35
Update	4,296.05
Delete	35.51
Insert	47,415.40
Select	14,983.69
Update	28,712.53
Delete	179.40
Insert	252,770.01
Select	85,014.17
Update	137,056.27
Delete	814.83

Test 6: Comparison of Results

	Insert	Select	Update	Delete
Image-4k	5,020	9.36	3,850	38.43
Image-8k	4,354	8.80	3,605	31.30

	Audio-Insert	Audio - Select	Audio-Update	Audio-Delete
Audio-4k	31,026	11,738	3,225	160
Audio-8k	28,439	9,841	3,148	114

	Video-Insert	Video - Select	Video-Update	Video-Delete
Video-4k	115,554	36,697	72,991	477
Video-8k	103,855	33,752	56,688	343

Test 6: Summary of Performance Improvement Overall

% increase	
Images	11.496%
Audio	11.088%
Video	15.968%

Test 6: Summary of Performance Improvement for Each Operation

	Insert	Select	Update	Delete
Images	13.28%	5.93%	6.36%	18.55%
Audio	8.34%	16.16%	2.39%	28.68%
Video	10.12%	8.02%	22.34%	28.08%

Test 6: Performance Improvement for Images

		Insert	Select	Update	Delete
Small	avi	17.31%	14.23%	0.67%	28.96%
	mov	34.20%	17.81%	1.97%	30.65%
	rm	28.18%	14.82%	22.15%	27.70%
Medium	avi	31.70%	26.75%	5.15%	31.18%
	mov	4.12%	17.55%	1.43%	34.81%
	ram	29.29%	19.89%	25.52%	20.19%
Large	avi	1.94%	5.48%	29.78%	28.28%
	mov	9.14%	7.52%	9.37%	24.27%
	ram	12.44%	2.10%	9.55%	31.41%

Test 6: Performance Improvement for Audio

		Insert	Select	Update	Delete
Small	wav	2.09%	1.06%	0.21%	12.23%
	mp3	39.58%	31.72%	24.16%	6.04%
	au	2.73%	6.86%	24.32%	21.19%
Medium	wav	16.54%	18.59%	5.34%	18.17%
	mp3	14.15%	10.37%	0.46%	21.12%
	au	0.55%	12.76%	2.86%	37.13%
Large	wav	6.68%	1.49%	3.05%	39.91%
	mp3	0.33%	18.72%	2.09%	33.38%
	au	20.88%	36.46%	15.27%	13.83%

Test 6: Performance Improvement for Video

		Insert	Select	Update	Delete
Small	Bmp	0.45%	21.44%	24.17%	2.96%
	Gif	20.26%	0.33%	32.76%	3.58%
	Jpg	0.09%	4.93%	15.43%	3.37%
	Png	5.90%	5.21%	5.49%	6.40%
	Tif	34.33%	5.02%	21.18%	3.86%
Medium	Bmp	30.58%	6.82%	6.29%	1.62%
	Gif	34.87%	6.93%	3.91%	6.05%
	Jpg	29.63%	3.53%	0.28%	17.08%
	Png	16.37%	5.19%	4.10%	18.12%
	Tif	33.63%	10.24%	9.14%	4.32%
Large	Bmp	2.81%	6.21%	6.27%	32.49%
	Gif	31.89%	1.74%	9.02%	27.28%
	Jpg	18.99%	1.71%	1.52%	32.85%
	Png	3.37%	4.77%	7.00%	21.48%
	Tif	31.92%	1.63%	6.26%	24.73%

C.5 MORD is Scalable

Test 8:Table showing the average time taken for Images using Dataset 1

		Insert	Select	Update	Delete
Small	Bmp	180.920	10.620	129.720	22.800
	Gif	255.920	10.020	143.100	22.820
	Jpg	204.380	10.320	144.680	22.860
	Png	1,118.460	8.460	1,070.020	22.480
	Tif	174.680	10.260	139.040	22.520
Medium	Bmp	786.560	9.720	574.980	28.720
	Gif	551.880	8.740	576.260	28.120
	Jpg	579.980	8.760	740.620	24.080
	Png	4,160.320	8.420	3,999.100	25.000
	Tif	567.860	8.140	701.220	24.380
Large	Bmp	5,218.440	9.040	4,868.120	61.880
	Gif	6,809.380	8.420	4,950.300	38.140
	Jpg	6,026.560	8.460	5,600.300	39.360
	Png	35,116.880	8.760	30,683.420	38.420
	Tif	6,299.720	7.800	5,585.020	57.820
	Average	4,536.80	9.06	3,993.73	31.96

Category Averages	
Insert	386.87
Select	9.94
Update	325.31
Delete	22.70
Insert	1,329.32
Select	8.76
Update	1,318.44
Delete	26.06
Insert	11,894.20
Select	8.50
Update	10,337.43
Delete	47.12

Test 8: Table showing the average time taken for Audio using Dataset 1

		Insert	Select	Update	Delete
Small	wav	829.800	143.700	284.300	29.800
	mp3	737.500	142.100	226.700	28.000
	au	351.900	90.600	76.400	26.000
Medium	wav	4,351.500	848.500	126.600	36.000
	mp3	2,732.800	737.400	1,153.200	32.900
	au	1,817.100	596.700	85.800	32.900
Large	wav	103,818.700	29,156.400	185.900	371.800
	mp3	32,361.100	10,792.300	14,725.100	406.200
	au	53,048.200	12,468.800	88.900	119.200
	Average	22,227.622	6,108.500	1,883.656	120.311

Category Averages	
Insert	639.733
Select	125.467
Update	195.800
Delete	27.933
Insert	2,967.133
Select	727.533
Update	455.200
Delete	33.933
Insert	63,076.000
Select	17,472.500
Update	4,999.967
Delete	299.067

Test 8: Table showing the average time taken for Videos using Dataset 1

		Insert	Select	Update	Delete
Small	avi	4,634.600	277.500	2,406.300	36.300
	mov	4,250.000	234.400	2,161.000	31.400
	rm	4,506.100	263.200	6,387.500	40.600
Medium	avi	41,023.400	237.600	10,436.000	60.800
	mov	11,031.100	235.000	9,201.500	59.300
	rm	17,329.800	232.400	40,365.700	231.300
Large	avi	263,228.200	228.400	228,412.400	1,153.200
	mov	205,828.100	232.200	85,834.400	340.400
	rm	158,226.600	231.900	45,107.800	325.100
	Average	78,895.32	241.40	47,812.51	253.16

Category Averages	
Insert	4,463.57
Select	258.37
Update	3,651.60
Delete	36.10
Insert	23,128.10
Select	235.00
Update	20,001.07
Delete	117.13
Insert	209,094.30
Select	230.83
Update	119,784.87
Delete	606.23

Test 8: Table showing the average time taken for Images using Dataset 3

		Insert	Select	Update	Delete
Small	Bmp	211.800	13.360	137.365	26.495
	Gif	207.025	11.875	142.115	27.195
	Jpg	204.535	11.880	144.950	45.550
	Png	972.815	12.815	726.550	26.105
	Tif	175.310	11.955	142.730	27.095
Medium	Bmp	1,276.555	11.875	580.140	31.240
	Gif	877.195	11.640	581.520	27.335
	Jpg	582.890	11.565	616.500	32.655
	Png	5,120.155	11.565	5,081.670	29.030
	Tif	1,187.745	11.870	702.555	29.075
Large	Bmp	7,571.950	11.720	5,205.320	53.245
	Gif	7,377.090	11.560	4,815.095	45.755
	Jpg	8,720.470	11.490	5,264.990	60.605
	Png	35,871.360	11.485	32,929.040	51.060
	Tif	9,893.045	11.650	15,699.680	59.910
	Average	5,349.996	11.887	4,851.348	38.157

Category Averages	
Insert	354.297
Select	12.377
Update	258.742
Delete	30.488
Insert	1808.908
Select	11.703
Update	1512.477
Delete	29.867
Insert	13886.783
Select	11.581
Update	12782.825
Delete	54.115

Test 8: Table showing the average time taken for Audio using Dataset 3

		Insert	Select	Update	Delete
Small	wav	1967.84	286.26	192.2	28.76
	mp3	1149.36	200.94	311.56	32.18
	au	1120	186.88	84.06	27.18
Medium	wav	16138.74	1589.06	134.06	45.62
	mp3	8619.06	1507.52	2325.62	56.56
	au	10542.2	1089.38	87.82	39.36
Large	wav	148949.38	42822.84	177.5	391.26
	mp3	99735.32	33716.84	23127.46	319.38
	au	79715.6	16915	87.5	195.62
	Average	40,881.94	10,923.86	2,947.53	126.21

Category Averages	
Insert	1412.4
Select	224.6933
Update	
Delete	195.94
Insert	11766.67
Select	1395.32
Update	849.1667
Delete	47.18
Insert	109466.8
Select	31151.56
Update	7797.487
Delete	302.0867

Test 8:Table showing the average time taken for Video using Dataset 3

		Insert	Select	Update	Delete
Small	avi	19,463.74	144.06	3,933.76	45.94
	mov	39,165.94	135.76	3,870.92	43.12
	rm	19,966.56	133.22	7,323.46	42.62
Medium	avi	49,268.72	45.30	22,967.18	134.04
	mov	45,499.66	45.30	26,000.00	154.98
	rm	45,150.92	46.24	35,388.44	198.16
Large	avi	565,160.84	146.90	211,388.16	1,155.92
	mov	563,201.28	126.26	210,065.60	451.22
	rm	563,200.02	146.90	210,065.60	503.78
	Average	212,230.85	107.77	81,222.57	303.31

Category Averages	
Insert	26,198.75
Select	137.68
Update	5,042.71
Delete	43.89
Insert	46,639.77
Select	45.61
Update	28,118.54
Delete	162.39
Insert	563,854.05
Select	140.02
Update	210,506.45
Delete	703.64

Test 8: Comparison of Results

	Insert	Select	Update	Delete
Total Image - Half	226,840	453	199,686	1,598
Total Audio - Half	222,276	61,085	18,837	1,203
Total Video - Half	788,953	2,414	478,125	2,532
Total Image - Normal	438,712	949	365,233	3,286
Total Audio - Normal	684,597	241,995	78,695	2,842
Total Video - Normal	2,816,564	843,810	1,417,583	8,737
Total Image - Double	1,069,999	2,377	970,270	7,631
Total Audio - Double	2,044,097	546,193	147,377	6,311
Total Video - Double	10,611,543	5,389	4,061,128	15,165

	Insert	Select	Update	Delete
Average Image - Half	4,537	9.06	3,994	31.96
Average Audio - Half	22,228	6,109	1,884	120
Average Video - Half	78,895	241	47,813	253
Average Image - Normal	4,387	9.49	3,652	32.86
Average Audio - Normal	28,528	10,019	3,150	120
Average Video - Normal	112,663	33,752	56,703	349
Average Image - Double	5,350	11.89	4,851	38.16
Average Audio - Double	40,882	10,924	2,948	126
Average Video - Double	212,231	108	81,223	303

	Insert	Update
Small - Audio	1,370.41	310.69
Small - Image	331.44	232.57
Medium - Audio	6,302.77	843.13
Medium - Image	1,422.78	1,354.12
Large - Audio	77,909.91	8,295.24
Large - Image	11,407.15	9,370.31

	Insert	Update
Small - Audio	1,370.41	310.69
Small - Video	331.44	232.57
Medium - Audio	700.31	93.68
Medium - Video	142.28	135.41
Large - Audio	865.67	92.17
Large - Video	114.07	93.70

C.6 The Granularity of MORD's Dataset can be Altered

Test 9: Table showing the average time taken by each Image format

		Initialize	Upload	Set Properties	Generate Thumbnail
Small	Bmp	11.46	131.86	47.86	601.04
	Gif	9.42	70.22	43.74	141.56
	Jpg	10.36	73.98	47.30	110.26
	Png	12.74	69.06	979.60	1194.80
	Tif	10.30	76.58	68.56	570.44
Medium	Bmp	13.26	661.70	80.84	761.30
	Gif	9.94	630.14	60.16	4674.68
	Jpg	8.92	710.94	63.26	328.12
	Png	14.52	613.50	3706.10	3946.88
	Tif	11.02	598.26	75.44	389.62
Large	Bmp	15.44	6864.58	169.76	7502.02
	Gif	10.28	9163.92	160.96	39901.96
	Jpg	10.38	12580.44	203.14	13651.32
	Png	12.68	6478.42	27537.24	246733.94
	Tif	9.72	6295.20	136.96	7143.76
	Average	11	3,001	2,225	21,843
	Without Png	11	3,155	96	6,315

Category Averages	
Initialise	11
Upload	84
Set Properties	237
Generate Thumbnail	524

Initialise	12
Upload	643
Set Properties	797
Generate Thumbnail	2,020

Initialise	12
Upload	8,277
Set Properties	5,642
Generate Thumbnail	62,987

C.7 MORD can be Extended

Test 10: Table showing the average time taken for Documents and other Test data

	Insert	Delete
doc	2075	44
pdf	1603	31
ppt	1128	31
xls	594	28

Category Averages	
Insert	1,350.050
Delete	33.500

Average Document	1350.05	34
Average Images	629.53	29
Average Audio	26834.87	84
Average Video	89644.33	164