The Development of a Parallel Database Environment for use with Corporate Geographic Information Systems

by Mette Tranter B.Sc., M.Sc.

This Thesis is Submitted for the Degree of Doctor of Philosophy
in the Department of Geography

The University of Edinburgh

May 1999

Abstract

This research proposes that the use of general purpose parallel architectures combined with parallel relational database technology provides a solution for rapid retrieval and analysis of geographic data in a corporate environment. Changes in corporate IT strategies have made the inclusion of parallel architectures viable, vendor support for parallel database software is evident, and research into parallel geographic information system (GIS) functions make commercial versions likely. To that end a substantial parallel GIS/database environment was developed to test the suitability of parallel processing for corporate GIS.

The parallel GIS environment brought together parallel relational database management systems and GIS applications on a parallel machine (Meiko Computing Surface) to investigate the ability of the system to support three corporate requirements: the ability to manage and analyse large volumes of data with reasonable response times, the provision of efficient access for GIS to integrated corporate databases, and the successful support of both transaction processing and longer, more complex transactions in the same environment. A series of performance tests were devised and split into two phases. Databases were constructed for each of the testing phases, including a substantial corporate database of some 15 million rows of real data.

Experimental results from the performance tests undertaken showed that the parallel system successfully supported both the GIS and parallel relational database management system software. It also provided insights into the use and configuration of the environment, the construction of complex queries using GIS/database interfaces, and the ability to support both transaction processing and complex queries in the a single environment. Finally, the implications and applications of the results are discussed.

Table of Contents

SECTION 1: BACKGROUND

SECTION 2: FUNCTIONALITY AND PERFORMANCE CONSIDERATIONS FOR PARALLEL DATABASES

SECTION 3: DEVELOPING A TEST ENVIRONMENT

SECTION 4: PERFOMANCE TESTS AND RESULTS

APPENDICES

List of Figures

List of Tables

Acknowledgements

This thesis has been completed due to the efforts, guidance and bullying tactics of many people over the years it has taken to complete. Although it is impossible to name them all, there are a number of people who have given their unstinting support during many moments of stress. I would like to extend a grateful thanks to the following individuals and organisations:

- to Richard Healey and Bruce Gittings, who were my supervisors throughout the research period. With suitable prompting, and despite a move down south Richard was a source of inspiration. I would like to thank him for his encouragement and enthusiasm. While left behind to carry the can, Bruce has bullied and harried me at every turn to complete this thesis. I would like to thank him for his perseverance and for his patience and help;
- to the Ordnance Survey, who provided the financial support throughout the research;
- to British Gas (Scotland) for the supply of data and documents which made the performance tests possible. I would particularly like to thank Murray McLundie for all his patience while I sorted out my supergrid pipes from my inlet valves;
- to the Department of Trade and Industry and all the industrial partners in the Parallel Applications Program who provided the machinery and software which made the research possible;
- to Smallworld, for their substantial technical support in setting up and managing a Smallworld application. I would particularly like to thank Dick Newell for his overwhelming enthusiasm and support for the project;
- to Mr Terry Pratchett, who kindly agreed to allow the streets of Ankh-Morpork to be incorporated into the large corporate database, and thus lightened the days spent creating large quantities of data for the data base;
- and to both Steve Dowers of the Geography Department, University of Edinburgh, and Scot Telford at EPCC, who each spent many a weary hour trying to solve technical problems at my insistence and remained cheerful and helpful while doing.

My final thanks must go to my partner Gavin who has supported me throughout the research project, and managed not to nag when progress was slow. His assistance is truly appreciated.

All trademarks are hereby acknowledged.

1. Introduction

1.1 The Move Towards Parallel Corporate GIS

The role of Geographic Information Systems (GIS) over the last fifteen years has changed dramatically from

> *"a Cinderella subject ... untouched by all but the most quantitatively avant-garde of Geography Departments..."* to *"...a multi-billion dollar industry and a major player within the broader field of information technology."* (Healey *et al.*, 1998a).

Initially, most GIS implementations were effectively pilot projects with limited data sets, and were often developed without detailed consideration of their potential linkages to wider management information system requirements within an organisation (Healey *et al.*, 1998b). However, it became very obvious that for many, particularly the utility companies and Government, a large percentage of the data underpinning their organisations was spatially referenced and these data was not being exploited to its full potential. To manipulate these data resources effectively, and to facilitate both strategic decision making and operational planning based on geographic information (GI), GIS technology would be necessary. This movement of GIS from the fringes of the business to its very core spawned *Corporate GIS*.

Corporate GIS is a term developed to describe GIS implementations within an organisational environment that play a central role in the IT strategy and are available to all those who have a need of spatial analyses. The term encompasses all levels of adoption from desktop GIS using off the shelf applications such as MapInfo and ArcView to bespoke systems consisting of various high performance hardware platforms and software applications.

The dramatic increase in corporate GIS projects focused attention on how to manage a GIS implementation successfully. There are two main elements to the introduction of corporate GIS into an organisation, the organisational implementation and the technological implementation. The organisational implementation, encompassing the culture and structure of the business, is of great importance and has received a great deal of attention in the literature over the previous 10 years (Openshaw *et al.*, 1990; Alla & Trow, 1990; Campbell, 1991, 1994; Padding, 1991; Campbell & Masser, 1992; McLaren & Healey, 1992; Mahoney, 1992; Sahay & Walsham, 1996, Peel, 1997a, 1997b). The challenge of introducing GIS into an organisation is in large part people-related and centred around building up an awareness and appreciation of the need for a GIS (Peel, 1997a). It is the attitude and support from within the organisation that will ultimately determine the success or failure of the implementation. It is often inextricably wound up in a complex process of managing change within an environment where uncertainty, entrenched institutional procedures and individual staff members with conflicting personal motivations are the norm (Campbell & Masser, 1991) and should not be underestimated. For these reasons the majority of literature available has concentrated on the success and failures of the organisational implementation, while assuming that shortcomings in the technology will be overcome at some point in the near future.

The technological implementation may take second place during the initial phases of implementation, but it is a **vital** part of the long term success of corporate GIS and should not be taken for granted in the haste to sort out other shortcomings in the organisation. If the GIS is unable to deliver all that is expected of it because it is slow, difficult to use, unable to produce the level of information required by the users in a usable form and unable to expand to meet the increasing demands made of it as users identify new applications and analysis, then it will simply not be used and the implementation will have failed. The evolution of GIS has shown the innovative ability of researchers, vendors and users to overcome a multitude of technological problems. These have encompassed the restrictions of limited processing power, mass storage of data, manual data input, and fledgling software (Healey *et al.*, 1998a) to become a technology of value to the business community, placing it at the heart of the organisation rather than as a novelty piece stored in the broom cupboard.

Of the two elements, the organisational implementation has been seen to be the province of the commercial world, and IT development that of the academics. However, as more demands are made of corporate GIS, users become more sophisticated, ever increasing levels of data become available, and technology such as the internet increases user numbers, then the focus of corporate GIS must shift from the introductory stages to concentrate on providing a sophisticated, user-oriented service that fulfils all the organisational requirements and expectations. To provide the service vendors, industry and academia will have to unite to embrace new technologies.

The very success of corporate GIS has exposed the need for a shift of focus from implementation to service provision. The introduction of GIS to the workplace has heightened awareness of spatial data available within the organisation to the extent that the volume of data, already large, is rapidly growing to unmanageable proportions. The storage facilities and analysis capabilities of GIS on such volumes of data are now scarcely adequate, even with the latest advances in processor speed and graphics capabilities (Healey, 1998a) and will really struggle to process the large datasets that are now becoming available. The advent of real time GIS applications (Xiong & Marble; 1996, Raper *et al.*, 1997), requiring sub-second responses on large, dynamic datasets, and the increasing use of GIS across the internet with its high levels of users and data traffic, will pose very serious IT problems for the business communities managing corporate GIS. These problems are looming and there is a real need to consider just how they are to be tackled.

Organisations will also have to provide support for increasing numbers of users, with a requirement to use large spatial data resources. These users will come from various sources and most will be unaware, and further will not care, how GIS provides a service, concerned rather with processing speed and response time. Carefully designed user front-ends to software hide underlying GIS applications from the user to the extent that the user is unaware that they are even using GIS technology. Examples can currently be found in the fire service with their emergency response units, which hold detailed plans of all large business premises, and in-car navigation systems. The internet is a major source of users expected to make huge demands of GIS. They pose serious problems because of the very large numbers of people it is possible to have requiring simultaneous access. Internet sites providing access to static documents such as government reports have been brought to breaking point by the sheer numbers of people trying to download files. Internet grid lock is already occurring and can only get worse as larger volumes of data are transported around the world. Therefore it is important that large volumes of data can be moved quickly from one place to another to reduce the time spent clogging up various networks.

One solution to these corporate GIS problems is the use of parallel computing. Parallel computing and parallel applications already available, such as parallel relational database software, can provide many of the facilities that current serial technology lacks.

Firstly, they can significantly improve the performance of large, computationally heavy, or data swamped tasks. The availability of multiple processors provides the ability to distribute the individual components of a large problem across a number of processors. This spreads out the work load, resulting in an increased throughput and quicker response time, than that of a single processor (Nicol & Willard, 1988). This ability will allow GIS applications to tackle the larger datasets while maintaining or improving the response times users have come to expect. The multi-processor environment, as well as manipulating much larger datasets, can also increase the capacity of the system by supporting far larger numbers of users. The users can be spread across a number of processors along with their tasks, again spreading the work load across the computing platform and improving response times for individual users.

The current move towards distributed computing is supporting this. The distributed or clustered system is connected by a network, such as ethernet, to provide a parallel (MIMD)[1] platform. Clustered computing is very attractive to industry because it makes use of resources that may spend a considerable amount of time standing idle otherwise. For example, by linking the workstations together large batch jobs can be processed more quickly. Although inferior in performance to the '*tightly*' coupled MIMD machines, network technology is improving, with Asynchronous Transfer Mode (ATM) technology capable of transfer rates in excess of 155 Mbytes per second. With support for such configurations reaching maturity, and some vendors marketing workstation clusters as dedicated parallel platforms, the boundaries of what constitutes a parallel MIMD machine are becoming blurred (Sawyer, 1998). Industry is already beginning to take a few steps down this road and those steps taken towards true parallel computing are becoming shorter.

Secondly, it is known that there exists a point where the fundamental limits to the speed-up of serial processors must be reached, even though new X-ray lithographic methods of etching chips are pushing this point further away. Regardless of when this occurs, any performance gains achieved in serial processors are multiplied substantially when the processors are run in parallel. Therefore, parallel platforms will always be able to outstrip the performance of the best of the serial processing machines. Thirdly, parallel computers are fully scalable, allowing the corporate IT systems to adjust and expand as the organisational requirements increase, while maintaining good response times for users. This also allows the size of corporate investment to be incremental. Fourthly, the move towards the use of commodity processors in parallel computers, not only makes them more maintainable and reduces their cost (Healey, 1998a), it neatly bridges the gap between the accumulated store

[1] Multiple Instruction Multiple Data (MIMD) computer is an example of a massively parallel computer. A more detailed description can be found in chapter three.

of serial applications upon which the organisation depends and the development and acquisition of new applications designed to take full advantage of the parallel environment. The final point to be made is the increased interest in parallel GIS applications and the work that has begun to define suitable algorithms (Healey *et al.*, 1998b).

In the last few years the practical application of parallel processing in corporate IT has become both accessible and visible due to symmetric multi processing (SMP) machines like the DEC VAX 6000-340 and now DEC Alpha, Sun Servers and Sequent Symmetries. These machines contain a small number of reasonably powerful processors that can either be used together, in parallel, to improve the performance of a single program, or used as individual processors to increase the throughput of a number of sequential programs. This configuration of processors permits the utilisation of existing code and applications which form the IT bedrock of an organisation, with little or no modification. The next obvious move is to more massively parallel systems, supplying a mixture of custom developed parallel software and existing serial software, where the benefits of parallel applications run on a greater number of processors can be felt. The ability to provide support for both these types of application has increased the appeal of massively parallel machines for organisations wishing to upgrade their hardware and software.

Over the last decade the focus of corporate GIS has been fixed on ensuring the successful implementation of software within the organisation. However, the time has come to move beyond implementation considerations and concentrate on the corporate GIS challenges of the foreseeable future. These involve, as mentioned above, the manipulation of very large spatial datasets, a greatly increased number of users, challenges posed by interactive GIS services over the internet, and real time GIS.

1.2 Exploiting Parallelism

Research into the use of parallel computing in Geographic Information Systems (GIS) spans at least ten years, but has for the most part concentrated on exploiting parallelism to improve the speed and efficiency of particular algorithms, or the use of custom built parallel databases, with limited functionality and capacity, to improve the retrieval of data. However, it had only been possible since 1992/1993 to construct a parallel GIS environment from hardware and software available *over-the-counter*. This development is of great interest to all investors in corporate GIS because it has the potential to provide solutions to a number of IT limitations that are becoming very evident.

The appearance of parallel architectures such as the Meiko Computing Surface (MCS) and commercial parallel relational database management systems (RDBMS) is so recent that little is known of the performance characteristics of this combination of hardware and software, and benchmark statistics from official bodies such as the Standards Council are few and far between. Further, the lack of literature available on designing benchmarks for GIS in a parallel environment, particularly on what to measure and how to go about it, means there are no agreed rules and regulations for comparing the performance of parallel GIS environments against either existing serial environments or environments using other parallel hardware and software.

Benchmarking and performance testing play a very important role in both the public and private sector when purchasing new systems and software, because the expenditure has to be justified not only to the senior management of an organisation but also to shareholders and government (where appropriate). Therefore, there is a need to establish whether there is a requirement for a parallel GIS environment in corporate computing, and then investigate the establishment of a series of calibrated performance tests to establish whether the parallel GIS system would be suitable as a corporate GIS engine. The research in this thesis intends to take a number of initial steps towards addressing these issues.

1.3 Aims

The research project aims to investigate alternative approaches to the integration of GIS and corporate database management systems in a parallel environment. Particular emphasis is placed on the identification of potential commercial benefits, but these must be balanced by consideration of technical and organisational risk factors.

The research project includes:

a) Interfacing of GIS software running in serial mode with a parallel database management system.

b) Interfacing of GIS software running in serial mode with databases mounted on both massively parallel and symmetric multi-processor architectures, linked via Fast Digital

Data Interface (FDDI) over optical fibre connections. This will allow evaluation of a multi-platform *data centre* type configuration.

c) Examination of the above in both *user parallel* and *application parallel* modes.

- With *user parallelism* the database is exploited to support a high throughput of individual users, each assigned to a particular database instance on a single node. Some of the transactions may be generated from within the GIS, but many others may be external to it, as in the case of management information systems (MIS) type queries. All database instances can access the same database on disk.

- In *application parallel* mode individual database queries are broken down and the components distributed across multiple database instances on different nodes, to optimise the response times for complex searches involving large data volumes.

d) Examination of the potential for storage, manipulation and retrieval of digital cartographic data using normalised structures within the parallel database.

e) The preliminary development of performance testing methods.

The alternative approaches and interfacing techniques are investigated using two databases. The first, a small pilot database is used to determine the working of Oracle on the MCS, GIS applications and their communication links. The second, is a substantial demonstrator database based on Cumbernauld, one of the Scottish New Towns. The basic data layers are derived from 100 1:10,000 scale Ordnance Survey digital map sheets, together with utility data supplied by British Gas (Scotland). The utility data were provided in raster scanned format for the cartographic information and as standard data tapes for the attribute information from the MINE database, held on a mainframe computer. The MINE database is a database developed and owned by British Gas and contains detailed information about every existing gas main. Gas mains are accessed using unique mains identifiers.

1.4 Overview of Project Stages

The research project was broken up into a number of stages:

a) Installation of compatible versions of the Oracle RDBMS on the Meiko Computing Surface and the VAX 6000-340 symmetric multi-processor.

b) Identification of requirements for Corporate GIS that could be addressed using parallel processing technology.

c) Design of demonstration GIS and its associated attribute/MIS database.

d) Conversion of raw digital or master data into GIS/database format so it could be used by different applications.

e) Testing of appropriate interfaces between Oracle and the different GIS applications, on the MCS and evaluation of alternative communication pathways (ethernet/TCP-IP or Meiko CSN). The first step was to determine whether technical problems existed and if so how they might be circumvented. The second step was to examine the extent to which the available interfaces could support multi-instance queries for *application parallel* as opposed to single instance *user parallel* queries.

f) Design of two performance test harnesses, for Oracle and Arc/Info, to collect performance statistics for the whole of the system rather than for individual components or single nodes.

g) Design of a test suite to evaluate performance across a wide range of functionality, including combined and GIS and TP type access to the parallel database.

h) Construction of the test suite.

i) Profiling of a range of operations run against the demonstrator system.

1.5 Analysis and Reporting of Results

For practical purposes the research was split into two phases. The first phase of the research was concerned with setting up the experimental parallel GIS environment to ensure the individual components would work together in the same environment and that the database links were functional. A suite of tests were designed for this phase to evaluate the performance of the database, under a number of different conditions, in its normal state. A performance test harness was devised to collect performance statistics from across the whole of the environment as well as from individual components. Results from the performance tests were stored in Unix files, and later transferred to Microsoft Excel for further analysis.

The second phase has concentrated effort on the interfaces existing between the GIS applications and the parallel version of Oracle RDBMS. Using the large demonstrator database for both transaction processing and large and complex queries, a range of functions have been examined. A second suite of performance tests were devised to explore different methods of submitting SQL queries generated from the GISs to the database to examine effects on performance. SQL queries were also developed to provide both transaction processing and complex query loads on the database. These were submitted to the database using a number of different strategies, to test the suitability of the parallel environment for supporting both transaction processing and complex query loads simultaneously. A second performance test harness was developed to collect performance statistics from tests initiated from GIS applications.

Descriptions of the test environment and the two databases are located in chapters seven, eight and nine. Results and analysis of both phase I and phase II can be found in chapters eleven and twelve.

1.6 Organisation of Dissertation

The thesis is organised into thirteen chapters, which have been split into four separate sections to aid the clarity of the presentation.

The first four chapters cover the introduction and background to the research project, detailing the recent history of the involvement of corporate GIS and relational DBMS with parallel computer architectures. They consider the origins of these three separate strands, their progress to date and the pressures and developments that have made it possible to unite them in this research.

The second section contains chapter five and chapter six. Chapter five describes the development of parallel databases from the initial database machines through to database management software designed to run on massively parallel hardware. Chapter six is concerned with configuration and functionality issues to do with benchmarking. It provides an overview of existing benchmarks available during the lifetime of the research project, and details the elements necessary to consider when designing and constructing performance tests and benchmarking exercises. The chapter provides the rationale behind developing a suite of performance tests used in later chapters rather than adapting existing benchmarks. More importantly it lays down the ground rules for performance testing in the later chapters.

Section three consists of four chapters, detailing the development of the parallel GIS environment and the design of two databases for use in the performance testing process. Chapter seven describes in detail the Meiko Computing Surface, and the GIS interfaces for two mainstream GIS packages Arc/Info and Smallworld. Chapter eight describes the pilot database used for testing purposes in phase I of the testing program. This is a small database of artificial data required for tests to characterise the system and check that all the separate entities were functional. Chapter nine continues the database theme, and describes the substantial corporate database that was designed and built for use with phase II of the testing procedure. The purpose of this database was to provide facilities to examine the performance of GIS/database interfaces, the use of views for representing tables in the database and the ability of the system to support different transaction work loads from both MIS and GIS.

The final section contains chapters ten, eleven, twelve and thirteen, and presents the measurement environment and the results and discussion concerning the research project. Chapters eleven and twelve detail the results and analysis from Phase I and Phase II respectively, with the results from Phase I feeding into Phase II. Chapter thirteen draws the conclusions of the research together, and considers the implications of those conclusions for corporate GIS users, GIS vendors, parallel architectures vendors and database vendors. It details some of the advances in parallel architectures and software since the end of the research project and finally considers topics for future research.

SECTION 1

Background

Corporate GIS has become more attractive to organisations, both large and small, due to a combination of factors. These include: increasing business pressures forcing organisations to maximise use of all information available to maintain a competitive edge, a management shift away from a compartmentalised organisation to one of cohesion, communication and co-operation, and the IT revolution which has provided the capacity to manage and analyse the available corporate information at a more affordable price. However, corporate GIS is becoming a victim of its own success as organisations attempt to progress from the implementation of pilot projects to fully fledged, organisation wide information facilities. Users are becoming both more sophisticated in their employment of strategic and operational data and more demanding in their requirements, needing faster environments with improved analysis capabilities, on larger datasets, that are accessible to a much wider audience than was ever envisaged previously. Although IT capabilities are progressing rapidly, the demands of corporate GIS always seem to outstrip their abilities, as larger datasets are becoming more widely available and use of the World Wide Web increases.

Research into the use of parallel architectures and parallel processing techniques concerning GIS algorithms has shown that there are many benefits for GIS users. One obvious one is the ability to break down algorithms and datasets to run on a parallel machine to improve the throughput and response time for specific GIS functions. Conversely, this also allows much larger datasets to be analysed in an acceptable time frame because more data can be processed faster. Secondly, parallel architectures allow support for a much larger number of users because they can be spread across a number of processors so throughput is much greater and no single user causes a bottleneck. Although there are currently no parallel GIS applications commercially available, able to incorporate parallel processing techniques such as raster-to-vector conversion or polygon overlay into GIS functions, parallel computers constructed from commodity processors e.g. Alpha, SPARC or 386/486 processors, exist, which can run both existing software designed for a single processor only or specialist parallel software designed to take full advantage of the parallel environment. These machines are able to integrate with corporate IT strategies because they can run existing software in which the organisation invested both time and money, while providing an upgrade path to full parallel processing and much improved performance.

Changes in business focus have brought the need for high quality, reliable strategic and operational corporate information to the fore. Growing appreciation of its value has led to huge investments in corporate database management systems that are able to store and manipulate very large volumes of data. The systems need to be fast, robust and secure to provide suitable access to data while retaining flexibility to change with the organisation. To provide the required support for an organisation processing very large volumes of data daily (e.g. Utility companies processing bills), database vendors, such as Oracle, have spent much time and money researching and developing parallel database management systems to run on parallel machines, particularly machines like the Meiko Computing Surface (MCS) - constructed from commodity processors. Relational database management systems have proved particularly amenable to parallel processing on these machines because they are composed of similar operations that are applied to consistent data, and therefore, the work load can be successfully spread across a number of processors. The work load can be composed of multiple users accessing a single database or complex queries broken up into small sections and spread across a number of processors.

The improvements in database technology, particularly the speed of data retrieval, the ability to store many different data types, the robust security and recovery mechanisms, and the capacity to handle very large volumes of data are features that have attracted many corporate GIS users. The need to store and manipulate large volumes of co-ordinate data, linked to large comprehensive attribute datasets, has become more urgent as data providers make larger and more complex datasets available, and organisations, realising the potential of their in-house datasets, require the ability to perform spatial analysis on them.

The following three chapters contain a review of these three separate strands that comprise the key elements of this research, namely corporate GIS, parallel architectures and relational DBMSs.

2. Influencing Factors for Corporate GIS

Geographic information is an important piece in the Information Resource Management jigsaw - dealing with the creation, production, collection, management, distribution and retrieval of information (Antenucci *et al.*, 1991). The uptake of Geographical Information Systems as a business information tool has been increasing more rapidly since the early 1990s as private and public sector organisations search for the keys to remaining successful and viable operations.

Tools such as Management Information Systems - for managing all types of corporate data from the operational to the strategic, and Executive Information Systems (EIS) - providing simple front end, user driven access to data, have been in use for a number of years and have a history of success in organising and disseminating strategic information in a timely fashion. MIS and EIS are extremely useful tools for summarising information to make it more accessible and digestible to the management of an organisation. They are created and maintained by experts who bring together departmental datasets to provide the high level, up-to-date information required by an organisation. These information systems provide expertise in a specific corporate sphere, fitting snugly within the organisational IT strategy while requiring little internal co-operation and few organisational changes. The success of these information systems in providing timely information in a succinct format fuelled a need for systems where the user had much more control over the environment, the data, and the analysis. This would allow the user to explore and analyse data in a much more flexible manner, without having to be a programming expert.

GIS are a category of information system that provide just such an environment. Being essentially an end user technology, they are used by people who are often not computer specialists (Openshaw *et al.*, 1990) and hand complete control of data and analysis to the user. However, this type of information system brings a number of corporate challenges with it. The first challenge is that GIS will not slot simply into the existing information strategies of an organisation. This is because a GIS is a complex system: it is resource intensive - from both a processing (compute and I/O intensive) and storage view point and uses complex data structures with varying numbers of variable length data records, spread across potentially huge linked files (Gittings *et al.*, 1991; Healey *et al.*, 1991). Many IT strategies designed to support particular corporate functions e.g. transaction processing type functions, could not support the onslaught of resource requirements made by GIS, so the GIS did not perform to expectation and the other functions were disrupted. GIS also requires data from many different sources, often in a variety of formats (Healey *et al.*, 1998b), which are often either not available or too complex to translate.

The second challenge is the tremendous level of communication and co-operation required from the human elements of the organisation in order to make GIS function effectively and efficiently. GIS is usually described as an integrating technology because it is able to manipulate data from disparate sources provided they have some form of spatial link, however obscure. This ability is often quoted as one of the main benefits for an organisation. However, it also implies that the organisation is ready to participate in a data sharing culture and that its employees feel secure enough in the company, and in their own positions, to advance a policy that may otherwise be viewed as diluting control over resources and thus weakening their positions (Mahoney, 1992). Changing the way data are used in an organisation often has knock-on effects. To allow for the integration of information from all parts of an organisation it might be necessary to change its structure to accommodate this. Thus data integration may provide a route for integrating the organisation itself to allow for the improved communication and data exchange (Padding, 1991). In today's climate where response to market place changes need to be fast, the ability to share and integrate data across traditional boundaries becomes a critical measure of how well an organisation can compete and survive (Dhillon, 1992).

2.1 The GIS "PUSH" Factors

The rapid uptake of corporate GIS seen in the 1990s is a response to a combination of environmental and business factors. They made it worthwhile to invest in software that was known not only for its costly nature in terms of computing, data conversion and expert operation, but also for its need for high level organisational co-operation and communication.

Some of the influencing factors have been:

- the Chorley Report (Department of the Environment, 1987) (and the subsequent AGI report 10 years on) (Heywood, 1997)
- the changing financial climate
- the utilities - their successes and the effects of privatisation (Roberts, 1989).
- activity of GIS vendors - a move away from large, complex mainframe/ workstation applications to desktop GIS for PC, and the design of many spatial applications purposefully developed for the user e.g. pipeline GIS applications, NHS applications
- increased availability of digital data
    - OS data converted to digital format e.g. Strategi, and the development of new digital datasets e.g. Meridian
    - satellite imaging
    - lifestyle data, address lists and other corporate data
- the realisation of the importance of good, reliable data for all aspects of organisational planning from day-to-day decisions to long term planning.

The following subsections consider in detail the effect of the Chorley Report, the changing financial climate and the utility companies.

2.1.1 Chorley Report

The Chorley Report was an influential document resulting from a committee headed by Lord Chorley. The report was published by the Department of the Environment (1987) and examined many issues involving the use of GIS and GI. It brought GIS to the attention of many organisations through the publicity it received and highlighted a number of major difficulties associated with the introduction of GIS. Three recommendations from the report that prospective users of corporate GIS should heed were: that information should be regarded as a corporate resource; that a corporate strategy was essential to ensure that the project did not fly off at a tangent; and finally that there was a need for a realistic appraisal of the costs and benefits of investing in GIS (Mahoney, 1992).

The report also highlighted four GIS projects based on utility companies or local government where GIS was being implemented to improve services. These were: the Dudley Digital Records Trial; the Taunton Joint Utilities trial; British Gas South Eastern Region GIS implementation; and the GIS project undertaken by Wessex Water Authority. They were projects that not only pointed towards the internal organisational benefits of using GIS data but also the possibilities for sharing data between organisations to make the process of exchanging information required by legislation, more efficient.

It has to be said that the Chorley Report did not prevent many of the mistakes made during the implementation process and a number of issues identified by the report are still outstanding (Heywood, 1997). However, it was a good advertisement for both GI and GIS.

2.1.2 Changing Financial Climate

The most powerful incentives for the adoption of corporate GIS have been based on financial considerations: the need to cut costs to survive; the need to manage and integrate data more effectively to control management and asset expenditure; the need to comply with government legislation or targets to avoid penalty costs; and finally, the need to provide acceptable returns for shareholders. The requirements for cost effective management and profit maximisation hold true across industry and commerce, national and local government, utilities and statutory authorities (Callaghan, 1989). To be successful all areas of an organisation have to be competitive, constantly trying to exploit advantages that will produce additional profit. Failure to be cost effective and produce a satisfactory commercial return on the assets employed is likely to put an organisation out of business or attract heavy funding penalties (Callaghan, 1989).

There are two main methods of reducing costs: the first is to make the work force more productive (Clegg, 1992), and the second to reduce the number of staff, as staffing costs account for a large percentage of organisational costs. The advantage of corporate GIS for an organisation is that it can contribute to both these strategies. By managing the spatial assets, reducing maintenance costs and improving efficiency, it reduces the need for both duplication of data (Homer & Watson, 1992) and extensive management of labour intensive paper-based map libraries (Morgan, 1991) to either free staff time for other activities or allow the organisation to reduce the number of staff employed. There is evidence that drastic reductions in the work force, known as "down sizing", have been made partly possible by the efficiency savings GIS enabled, allowing staff to be deployed more rationally and staff time to be used more productively (Rauen, 1993), manage stock precisely and locate problems more quickly and accurately.

Turning organisational data into digital format and making them available as a corporate resource has increased accessibility and made infinitely more flexible than the "frozen image" of paper systems (Webb & Todd, 1991). The sheer volume of paper, the costs incurred in managing the paper datastore, and the complexity and time involved in extracting useful data from it fuelled the drive for conversion to digital format. The aim was to reduce operating costs, improve returns on assets and improve customer service, to make the data more accessible and more manageable, enabling better decision making (Fanner, 1997). However, the conversion process was not without its problems and could in itself be a financial constraint on the organisation (Denness, 1997). Despite its restrictions the paper map actually represented a highly efficient means of storing data. A very large number of co-ordinate pairs are required to describe the complexity of topographic features, together with information describing the relationships between these features (Dowers *et al.*, 1991). To convert it costs both time and money to transfer detailed information, and money to acquire sufficiently powerful hardware and software to store and manipulate the data.

External pressure provided further impetus for GIS implementation and data conversion. Privatisation of the gas, electricity and water companies (in England and Wales) and regulatory pressure of government bodies such as OFGAS and OFFER changed the focus of management from the supply of the product towards a much higher level of customer service. The utility companies have very strong financial incentives to invest in corporate GIS in order to integrate and co-ordinate their spatial data, because the success of their business hinges on geographically-based assets i.e. mains/cables and customers. Although, some of the service is still related to delivery, much more attention has to be placed on the *customer interface* (Hobson, 1992) to encourage customers to remain with a particular company, locate potential customers with the desired social and spending profiles and then persuade them with low product prices and evidence of high customer satisfaction to become customers.

The regulatory bodies also play an active part in creating financial incentives for corporate GIS by constantly regulating the price of the product. If they make a recommendation for either a price freeze or reduction savings have to be made through efficiencies, redundancies or expansion of business.

2.1.3 The Utilities

The importance of the utility companies in promoting the use of GIS as part of the corporate IT strategy can not be under-estimated. In Britain and America the utility companies and local government have pioneered many of the system experiences that are relevant to contemporary GIS development (Antenucci *et al.*, 1991). They were able to demonstrate significant business advantages and cost savings for managing spatial data using IT, and demonstrated that when properly managed, data conversion from paper to digital format could enhance access and make its use much more flexible. For example:

- Reduced operating costs through increased efficiency
- Better return on assets through increased product throughput
- Reduced or deferred capital expenditure
- Improved service to customers (Webb & Todd, 1991)

The success of utility based GIS projects (Ives, 1992) and local government usage (Campbell & Masser, 1992; Denness, 1997) have had a major impact on the subsequent spread of GIS pilot projects found running in many business sectors. These range from financial institutions (Hornby, 1990; Branagan, 1995; Nitsche & Reuscher, 1997) to the retail and leisure sectors (Callingham, 1995) to rural conservation (Rideout, 1992).

The utility companies were one of the first to see a potential for corporate GIS to manage their vast spatial databases. GIS brought the potential for integration of many information management systems both within the organisation and externally through joint data exchange programs with other utilities and local government. This was driven, in part, by legislation in the form of the Street Works Act (1950, 1991) which required the exchange of information pertaining to the laying of cables or mains. In a world where the utility companies were facing many changes including, privatisation, operating in a competitive market place, and working within financial constraints placed upon them by regulatory bodies, GIS offered a means of managing change. GIS also provided a solution to manage the over stretched paper information systems that required a large, dedicated work force to maintain them.

The initial aims of the GIS projects were to increase efficiency within the organisation and the usability of the information. Improved spatial data handling presented many opportunities to the utilities for efficiency savings, from the laying and repairing of mains and cables to enhancing customer support, from integrating internal systems to facilitating the exchange of data with other utilities and external organisations such as Local Government (Homer & Watson, 1992; Gadd, 1992; Webb & Todd, 1991).

The utility companies are in a unique position because their main business is based on assets that extend the length and breadth of the country, delivering essential services to almost every home and business. Between them the utility companies were responsible for over 1.65 million kilometres of underground mains in 1987 (Department of the Environment, 1987), a figure which is greatly increased due to the government-led building policy to make more housing available at an affordable price, the escalating demand for telephone lines (particularly second lines for use of the internet and e-mail), and the introduction of optical fibre networks. In 1987 the Chorley Report calculated that the mains and cable networks had a replacement value of over £117 billion, which will be substantially more in today's market. The largest spatially referenced databases held by the utilities relate to distribution networks, customer records and street-work notices and were therefore candidates for conversion and integration.

2.2 The IT Management Revolution

In the last fifteen years the IT landscape has undergone a series of radical changes. IT responsibility has, rather like a pendulum, swung away from a rather rigid centralised approach based around the mainframe computer, over to one based on departments with stand alone PCs and workstations, and then back again towards the centralised control of distributed networks of PCs and workstations. These shifts in strategy have been mirrored in the development and growth of corporate GIS.

2.2.1 Centralised Control (I)

IT management of the 1970s and early to mid 1980s was a product of the computer systems available to large organisations. The IT strategy was based mainly on large mainframe computers that served the needs of the whole organisation. It was maintained centrally and accessed through a network of dumb terminals. The GIS applications available reflected this structure. They were centrally managed programs, designed for specialist users, and command driven. Many of the GIS installations were bespoke applications which were good at their intended job but had poor integration with rest of the IT system. Many GIS functions had to be performed in batch mode overnight because processing time took hours, and crash recovery was extremely important because many of the systems were unstable.

Many of the reasons for the failure of GIS projects in these early systems stemmed from a mixture of inadequate technology, badly designed software and an inability to manage GIS projects effectively. Users were put off by the poor response times, the inflexibility of the system, the complex commands it was necessary to learn, and the lack of usable data (Openshaw *et al.*, 1990). Organisations were less than enthusiastic because their large mainframes and databases were set up for transaction processing and could not cope with the load GIS imposed. This was largely due to the large CPU and I/O demands the applications made upon IT, and was confounded by an inexperience of GIS requirements.

The final compounding factor was that the hardware peripherals and storage media were all very expensive, and often did not live up to the expectations raised by the vendor (Openshaw *et al.*, 1990). Quite often it was easier to return to the paper systems, which although cumbersome, could be an easier and faster option than the mainframe and could be much cheaper!

2.2.2 Departmental Control

The haste to move away from a centralised management style and its cumbersome centralised IT approach with large mainframes and dedicated transaction processing applications, pushed both management and IT responsibilities out to individual departments. Departments became almost autonomous units with responsibility for their own IT strategies, budgets and data management. However, the move to a departmental IT approach did not bring about the information revolution that was anticipated because there was no over-arching control of the IT strategy. Individual departments invested in hardware and software with little reference to the strategies of other departments. The result was a series of incompatible systems, a reduced level of communication, a protectionist attitude towards resources - due to the need to recover costs from other departments - and an unwillingness to co-operate with other departments to sort out problems.

The GIS applications developed for large mainframes were of little use in a departmental situation, requiring both specialist knowledge and expertise to operate them. They were slow at producing information and were inflexible, unable to respond quickly to the changing information requirements of a department. The departmental approach encouraged the development of small GIS projects based on departmental data, which although successful in their way, were very much stunted in growth due to restrictions on funding, expertise and access to organisation wide data. It also encouraged the development of more slim-line GIS leading, eventually, to the desktop systems available today.

The move to departmental management of projects and the use of work stations and PCs revolutionised the working environment and brought corporate GIS tantalisingly within the reach of organisations. IT did not require a battery of programming experts to use it, hardware and software were more flexible and costs were beginning to come down. However, another set of obstacles to corporate GIS became apparent and were based not on IT failure but the inability to define a GIS project, justify the cost to management, or successfully manage the project. The problem stemmed from the fact that most projects were justified on a purely financial basis. The following list sums up the frustrations for those trying to start a GIS implementation (Openshaw *et al.*, 1990):

- difficult to justify GIS by cost/benefit analysis only - many of the benefits are hidden or difficult to quantify in monetary terms. The reliance on cost/benefit analysis made it difficult to write a meaningful project structure;
- the structure and scope of pilot projects were not thought out sufficiently and there were no clear views of what the requirements really were;
- the scope was either:
    - too large: so much time and money were spent converting data for GIS use that the expected benefits might not materialise for several years. By that time the management had lost interest and withdrawn support;
    - too limited: the project was not representative of the real life system requirements so when it was scaled up it did not work properly e.g. response time was extremely poor which frustrated users;

- little or no support from higher up the management structure;
- hardware/software did not do what the vendor promised most faithfully that it would. There was no real comparison of different systems to find most suitable;
- scant GIS skills.

The uncoordinated approach to IT acquisition and data acquisition, and the rising costs of maintaining a multitude of different systems has swung the pendulum back towards centralised control.

2.2.3 Centralised Control - the sequel

The move back to centralised control began as the need to remain competitive became paramount and businesses, realising the benefits of all departments pulling in the same direction, attempted to combine resources from different areas of the business. This new centralised approach to IT, fuelled by changes in management and advances in technology, is not the restrictive control of the past, but rather an attempt to ensure organisational compatibility with extensive use of networks to link the business together. It is in this climate that corporate GIS has really begun to flourish.

Two organisational developments in particular are pertinent to the continued development and expansion of corporate GIS. These are the concept of a Data Warehouse and the rapid expansion and use of networks, in particular organisational intranets and the Internet, to provide a means of flexible communication and data sharing. They are of great significance because they identify roles and functions for corporate GIS within the organisation that can be clearly understood by both management and project teams and, therefore, move the justification process for corporate GIS implementation away from cost/benefit analysis.

2.2.3.1 Data Warehousing

Since 1997 the concept of the Data Warehouse has emerged. This is an information strategy that links together disparate systems and information sources from different locations, co-ordinates diverse data formats (often incompatible with each other), and integrates them in a central location known as the *warehouse*. The warehouse is accessible to a multitude of applications to exploit the resulting database, from a strategic level to day to day running of the business. GIS is one application that will benefit greatly from such a strategy. It has long been recognised that successful corporate GIS implementations require integrated data sources to produce the results expected of them, and that GIS is the catalyst for that integration process. However, this has been proved problematic because GIS is usually viewed as a software application rather than an information strategy and has therefore struggled to reach its full potential because of the lack of will to change without senior management clout. Data warehousing frees GIS from the burden of the role of data integrator, allowing it instead to concentrate on producing results.

Secondly, data warehousing is important because it provides a corporate framework for GIS to sit within. In the past it has not always been clear to managers just how GIS could fit into their organisation and what benefits it could bring them because they have relied on 'Cost/Benefit' analysis (Ives, 1992; Lodwick & Cushnie, 1990; Webb & Todd, 1991). This, as is mentioned in the previous section, is a mechanism through which it is very difficult to justify the expenditure for GIS because while the costs are patently obvious, many of the benefits are of the 'intangible' variety and therefore can not be quantified. Data warehousing goes some way to solving these perception problems by giving GIS visible roles as both a data provider to the warehouse, contributing value-added spatial data, and as a data consumer application using the combined data available. Data warehousing is a very valuable strategy because it can place corporate GIS at the heart of the IT strategy making it a truly organisation-wide tool, while boosting the business case for organisation-wide implementation.

2.2.3.2 Organisational Intranets and the Internet

The rapid development of network based communication through intranets, extranets and the internet have had a very positive effect on the desire to share information at both a department and individual level. The change in attitude has come about for several reasons. Firstly, both staff and departments can quickly feel the benefits of data exchange using web-enabling technology, because others reciprocate and provide access to data they manage. Secondly, encouraging the publication of data through the web does not reduce or weaken the position of those who produce and manage it. Rather, it tends to strengthen their position as more people become aware of the resource and depend on the accuracy and reliability being maintained. Finally, software suites such as Active Intranet make the publication of data a simple process and encourage individuals and groups to take possession and become responsible for their own areas of the organisational intranet. By giving individuals a vested interest in sharing and maintaining information, web technology is beginning to succeed in developing an organisational culture built on facilitating communication rather than stifling it.

This is beginning to have an impact on corporate GIS. Not only is it making many more data sources available for research and analysis, it is providing a means for publishing results, and further, providing interactive access to spatial data. It also provides an excellent medium for providing access to a wide variety of GIS front-end applications that are simple to use by anyone familiar with a mouse, are user-friendly, and gently guide the user through a series of choices to the final result and with acceptable response times. These applications can be designed such that the user is not even aware that the underlying software driving the front-end is GIS.

The internet opens up access to spatial information and analysis to a much larger audience making it a truly corporate information system. With access to GIS widened through the use of internet/intranets, it allows many new types of application to be built e.g. real time mapping, making it available minute by minute, anywhere in the world. On a world-wide scale it could prove very useful in disaster planning and management. Closer to home, gas meter readers could use the internet to control their daily work from down-loading details of the next job, to shortest route planning, to traffic information, to the constant recording of their position making deployment easier.

However, there are still many difficulties to be overcome before web technology provides the perfect solution. With the increasing use of real time mapping and analysis in conjunction with the World Wide Web (WWW), and the existence of millions of potential users that the web enables, GIS web activity will very soon result in enormous demands for multi-streamed performance. Already many web-servers are receiving tens of thousands of accesses every day. As the sophistication of these accesses develops, with demands for complex database queries and the mapping of results, so the performance of the systems servicing those requests must be able to react rapidly to satisfy a user community at present frustrated by lack of network band width. Websites such as Digital's Alta Vista search engine use powerful, multiprocessor servers to satisfy the current level of demand - which consist of simple and unsophisticated queries. It can not be long before the demands made of

GIS software, customised into vertical markets such as tourist information systems, reach and rapidly exceed those levels (Healey *et al.*, 1998b). Software is currently available to launch GIS onto the web through applications such as AutoDesk, ArcView and MapInfo.

The availability of GIS web technology is beginning. Maps on the internet are not a new phenomenon: in 1997 there were approximately 200 sites with bit map images that could be viewed and queried using standard on-line browsers (Ireland, 1997). Using vector data in a similar fashion is very attractive as it would make the maps dynamic and reduce network transactions.

2.3 Computing Developments

2.3.1 Hardware

A tidal wave of innovation, driven by advances in PCs and compounded by the Internet, has broken over the IT scene (Barr, 1998). Moore's Law, that computer power will double every eighteen months, is still very much applicable. Advances in PC systems are such that the top end PCs overlap significantly in power with the bottom end of the traditional UNIX workstation market - at a much lower price (Barr, 1997).

Advances in processing power, memory, storage technologies, networking (including the Internet) are making GIS more widely accessible to businesses. Improved manipulation of graphics through standards such as OpenGL, and PC graphics handling using the Accelerated Graphics Port (AGP) (Toon, 1997), in conjunction with hardware and networking advances, have developed a demand for desktop GIS as demonstrated by a survey of GIS in the business sector (Grimshaw, 1997). The falling price of PC technology has ensured that there are few businesses that do not have a local PC network of their own and there are few employees who do not routinely use computers as part of their job.

2.3.1.1 GIS Limitations

Despite all of the advances in PC and processor technology there are still significant limitations to corporate GIS. Power on the desktop has brought excellent interactivity to the user interface, but has yet to be harnessed enterprise-wide for cost-effective GIS processing

(Healey *et al.*, 1998a). Processing, memory, disk and network constraints still abound, forcing a compromise between the performance of a corporate GIS and the size and complexity of geo-datasets used. This is exacerbated by contract deadlines for corporate GIS projects that are too tight to accommodate the processing time required to manipulate or analyse large geographic datasets. Even with the various technological advances which have gone some way to meeting the increased demand for processing, performance will still eventually be limited by the speed at which millions of electron switches contained within a processor can be operated (Sawyer, 1998). Predictions vary, but most estimate that within the first quarter of the 21st century further increases in the performance of a single processor will be impossible.

The total costs of ownership of PC networks are only now being realised. Powerful but inexpensive PCs still require substantial system support, which has become increasingly complex and expensive because it is spread over numerous distributed machines rather than a few centralised servers, with little support for standard configurations. It is apparent that in a corporate environment the support costs for PCs exceed their original purchasing price every year of their life (Barr, 1998).

2.3.1.2 Networking Developments

There are a number of initiatives to develop less costly environments, two of which are being designed by Oracle/Sun Micro Systems and Microsoft. The Network Computer (NC), supported by Sun and Oracle, is a device with limited local processing and storage capacity which will act as an intelligent client to a server. It offers the relative ease of use and interactiveness of a PC, but without the support costs. Similarly, the *'zero administrative cost'* initiative from Microsoft transfers the costs from supporting individual machines to providing facilities on servers to maintain each machine remotely. In either case the role of the client computer is much reduced and has become known as a thin client (Bytes Technology Group, 1997). The desire for thin clients has been further fuelled by the aspirations of information providers on the World Wide Web, particularly those who would like to provide *active* content, i.e., content that users can interact with rather than just view. It is highly feasible that parallel processing will have a strong part to play in these configurations providing parallel server capabilities to serve thin clients.

These developments all make IT and computing more accessible to all employees in the organisation, allowing very sophisticated applications to manipulate larger volumes of data than were ever dreamed possible in the early 1990s. However, these advances in IT have created problems as well as solving them, raising user expectations about the levels of data that can be manipulated and the speed of processing, and have created networks that are costly in both time and money to maintain.

2.4 Future Moves for Corporate GIS

While the market place demand for corporate GIS is expanding and changes in both organisation management and IT structuring are beginning to facilitate the adoption of corporate GIS, there are still many challenges for corporate GIS to overcome. For the foreseeable future those challenges are of the technical variety, for example: processing very large data sets at an acceptable speed; supporting and managing large user groups; real time GIS and web-based GIS. One solution to these difficulties is parallel processing, using multiple processors to speed-up the processing and analysis of very large datasets, or spreading large numbers of users over many processors.

The technology of symmetric multi processing, be it with Pentium, SPARC or Alpha processors, has gone some way towards this, bringing about a quiet revolution in the file servers of numerous academic institutions and commercial organisations. The number of processors are usually much smaller than those in massively parallel machines, often between 4 and 16 processors. However, the gains in throughput are considerable (Healey, 1996).

The move towards centralised servers and thin client technology outlined above also supports a move to parallel processing. By managing PC environments through a centralised server or small group of servers using thin client technology, there is a ready-made place in the IT structure for parallel servers to provide the processing power and network speed required to process and analyse large data sets. The advantage of this strategy is that by locating the parallel server at the centre it insulates the main body of users from the complexities of parallel processing, while providing a much improved service. If demand for parallel architectures increases sufficiently, it should also provide the necessary impetus for GIS software vendors to take full advantage of the technology and consider the implementation of parallel algorithms and data storage.

These developments are highly significant for parallel computing in both the GIS and business communities. The uptake of parallel computing, particularly in the business community, has been slow due to the rather chequered history of failed hardware companies and unsatisfactory hardware tools (Healey, 1996). The literature has for some years been drawing attention to the promise of general-purpose parallel computing (Hack, 1989) and its ability to distribute components of a large computational task across a number of processors producing (theoretically) more rapid throughput than that of a single processor (Nicol & Willard, 1988). The trend towards commodity processors, even if their interconnects are custom built, has made parallel machines more maintainable, more affordable and able to run versions of standard systems software such as UNIX (Sawyer, 1998).

3. Parallel Processing in GIS

In the GIS community it is recognised that the developments in high performance computing technology have had a significant impact on scientific progress (Ding & Densham, 1996). The importance of geo-processing is increasingly recognised and it has been suggested that parallel processing is one of the most significant trends in hardware for GIS (Dangermond & Morehouse 1987). Many spatial problems are inherently parallel (Armstrong, 1994). The majority of spatial analysis problems require substantial quantities of data, which are usually more readily broken down for parallelism than the processing tasks (Xiong & Marble, 1996).

There are a number of reasons why parallel processing is appropriate for GIS. Firstly, GIS operations are multi stage, requiring the application of several algorithms in turn - these algorithms may contain sequential components. Secondly, computation and I/O may be interleaved during the same operation - this may be due to the function itself or the very large size of the datasets - it is not safe to assume the datasets will all fit into memory. Finally, a GIS algorithm may require to link to a proprietary database manager to allow the storage and manipulation of co-ordinates or attributes (Healey *et al.*, 1998b). A large proportion of data required for analysis may be held in databases located externally to the GIS. Examples of such databases are those used to store finance, sales, marketing, and stock control information.

The increase in availability of both GIS and remote sensing data (Healey *et al.*, 1998a) and the trend towards real time GIS applications (Xiong and Marble, 1996) adding sub-second response times to analytical operations on large and highly dynamic datasets require greater processing power than is currently available. The need for new, much larger datasets, particularly in image processing will require not only more processing power in order to manipulate and analyse the images, but also new processing techniques to allow this to happen in a realistic time frame. There are significant steps being made in the business community towards the use of parallel servers to boost processing speed for an increasing user population as the demands of both the users and the software continue to increase. There are two initiatives of significant interest concerning GIS and parallel processing: data warehousing and thin client technology. Both these initiatives (as explained in Chapter two) require a powerful server, or a small network of powerful servers at their core to provide their respective services (Lee, 1995; Sybase, 1999; Wong, 1998). As a corporate resource, access to parallel processing for GIS can be made through these mechanisms. It also provides a solution to a number of problems that corporate GIS has been struggling with, by providing the processing power and storage capacity long sought after, while removing the burden of justifying the expense of specialist hardware for GIS use alone. It is therefore essential that, as these initiatives become more fully fledged, GIS is in a position to make the best use of the processing power available.

In this chapter, the applicability of parallel processing for GIS is explored. The chapter is divided into a number of sections describing the different classifications of parallel architectures, methods of measuring the performance of algorithms designed for parallel platforms, parallel geo-processing and finally some of the benefits that have attracted the business world towards parallel processing.

Parallel processing offers two benefits to GIS: the opportunity to investigate new ways of representing, manipulating, analysing and exploiting space and spatial relations; and greatly increased computational throughput (Ding & Densham, 1996).

3.1 Parallel Architectures

Serial computers have, for the most part, followed the *Von Neumann* model of computation, which describes the functionality of a computer in terms of a single processor, freeing users from questions as to the programming model of a particular machine and hardware designers from questions about the potential use of their designs. The simplicity of the model is attractive to designers of both hardware and software. Machines are simpler to build and simpler to program if they have one of everything (Sawyer, 1998). It has long been emphasised that a similar model is required for parallel computing although it has never been obvious how to implement the abstract computing models. However, in the last few years researchers have started to try to bridge the gap between theory and practice by providing high level models of parallel computation that can easily be implemented on current technology.

In 1972 Michael Flynn proposed a taxonomy for computer architectures (Flynn, 1972) as follows:

- **SISD** - single-instruction single data-stream. This is the architecture for the traditional serial computers that have only one program operating on a single set of data at a time.

- **SIMD** - single instruction multiple data-streams. In this architecture many processors simultaneously execute the same instructions on different data elements. This is the basis for the massively parallel machines, the three most significant being the AMT Distributed Array Processor (DAP), Thinking Machines Connection Machine (CM), and MP-1 from MasPar (Trew & Wilson, 1991). These machines use many thousands of simple processors and can achieve supercomputer performance on problems which involve little or no interaction between operations e.g. image processing.

- **MISD** - multiple instruction single data-stream. A machine of this nature would apply many instructions to each datum fetched into memory. To date no such computer has been constructed that fits strictly into this model (Trew & Wilson, 1991) and there are no applications that have required this architecture (Sawer, 1998).

- **MIMD** - multiple instruction multiple data-stream. A MIMD computer is an evolutionary step forwards from SISD computers. They contain a number of independent processors (normally more powerful than the individual processing elements in SIMD) each executing an individual program. There are several methods of building MIMD computers, depending on how the processors are linked together for communication and memory. Examples of MIMD computer are the Meiko Computing Surface, and the Cray Y-MP/832, Sequent Balance, Cray T3D.

Of these classes the SIMD and MIMD are relevant to parallel computing for GIS.

3.1.1 SIMD Machines

SIMD parallel computers consist of a number of identical processors, each with its own local memory where data and programs can be stored. All of the processors operate under the control of a single instruction stream which is issued by a master processor. The processors operate synchronously, that is, at each step all processors execute the same instruction, each on a different data portion (Akl, 1989). This type of architecture is particularly suited to problems that can be divided up into a number of similar sized parcels of work, each of which contain a similar volume of work, said to be inherently load-balanced.

Spatial data are complex but have unique characteristics that can be exploited for SIMD parallel processing. Firstly, spatial data are about particular locations and their attributes in space, and secondly, they contain information about spatial relations, for example, spatial correlation among different geographical objects at the same place and spatial auto-correlation over different locations (Xiong & Marble, 1996; Peuquet, 1988, 1994; Goodchild, 1992).

The sub-division of space is the main method used for representing locations. The study area is usually sub-divided into numerous small cells, mostly of regular shape, with attributes recorded for each of the cells. Data in this format will often naturally decompose along these spatial subdivisions (Xiong & Marble, 1996). For example, raster based processes such as image processing involve performing the same operations on each pixel of the image, such as taking a weighted average of its value and the value of the four nearest neighbours. If each pixel is mapped to a separate processor a SIMD machine can process calculations for each pixel simultaneously, producing the clean image in much less time than a serial machine could manage (Trew & Wilson 1991; Tomlin, 1990).

However, the restriction of executing the same instruction on every processor limits the applications that can be run efficiently on a SIMD machine (Sawyer, 1998). This has proven so for database applications. The multi-user, I/O intensive requirements of database processing do not lend themselves to this particular type of architecture and so SIMD computers have not gained acceptance as database machines (Healey *et al.*, 1998b).

3.1.2 MIMD Machines

MIMD machines consist of a number of processors working simultaneously on different data sets using different instructions. They are more general purpose and flexible than SIMD architectures (Ding & Densham, 1996). The basic model can be composed in many different ways. The main distinction made between the different types of MIMD computer lies in the power and numbers of processors. They range from those with a small number of powerful processors through to those with large numbers of less powerful processors, and of course all the stages in between. MIMD computers fall into three main categories.

3.1.2.1 Small Numbers of Powerful Processors

These have tended to evolve from existing computers. All the parallelism is produced by the compiler allowing the sequential code from other computers to run without adaptation. An example of this is the Cray Y-MP machine. The main restriction of this type of computer is that automatic parallelisation is only applicable to certain well defined problems and produces well below optimal results.

3.1.2.2 Moderate Numbers of Standard Microprocessors

The processors in these machines (e.g. Sequent, Alliant) are often attached to a single memory store by bus-based links. The machines can be constructed from a variety of different processors and, while custom-made processors are still being constructed, the majority of machines make use of commodity processors e.g. 386/486, SPARC or Alpha. This produces a computer that provides coarse grain parallelism.

Multi-processors that are constructed from commodity processors are a way of allowing existing applications to take advantage of parallel architectures with no extra effort on the part of the organisation/business. They are also commercially appealing because they provide a transition to full parallelism. The principal draw-back with this type of MIMD computer is that there is an upper limit on the number of processors that can be used with bus-based processor to memory links, although this restriction does seem to be reducing over time.

3.1.2.3 Large Numbers of Processors

Large numbers of processors linked together usually avoid the memory bottlenecks of the last two categories by giving some memory to each processor - known as distributed memory. The principal challenges with this approach occur because processors have to communicate with each other to locate data not stored locally (EPCC, 1992). This is thought of as medium grain parallelism.

3.2 Memory

There is a further classification of parallel computers based on how the processors are able to access memory. With single processors all memory has to be addressable by the processor. With multiple processors there is a choice between allowing each processor access to the whole memory and allowing each processor access to a certain part. The former is known as shared memory, the latter is known as distributed memory. Both styles have advantages and disadvantages.

3.2.1 Shared Memory

One natural approach, when using multiple processors, is to give all the processors access to a single, global memory space, usually through a common bus. This is known as *shared* memory (see Figure 3-1).

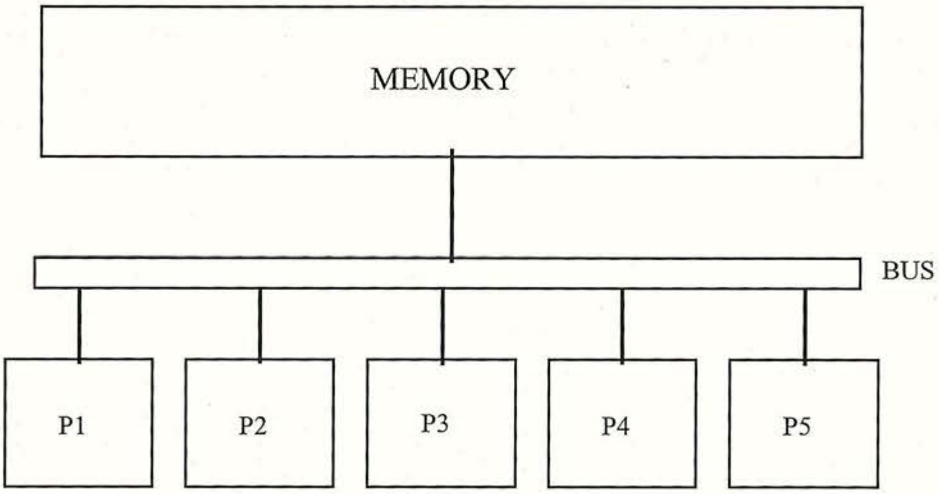


Figure 3-1: Typical Shared Memory MIMD Architecture (after EPCC, 1992)

In this arrangement processors communicate through objects placed in global memory. The single memory space brings the shared memory model closer to serial programming than the distributed memory model and is a very stable paradigm. Its attractiveness is that it is relatively simple to program because only the algorithm is parallelised and the data are left undivided in global memory (Fox *et al.*, 1988). With sophisticated compilers, applications are relatively easy to run on MIMD-SM computers - loop constructs in programs might be automatically shared across processors by the compiler. Most techniques developed for multi-tasking computers can also be used directly on shared memory computers.

However, there are limitations using this configuration. Firstly, performance is limited by contention for the bus (the greater the message traffic the more contention becomes a problem) and by problems with memory access control (which process should update an item of memory when two wish to do so concurrently). Secondly, synchronisation is required to ensure two processors can not simultaneously update a single memory area (this synchronisation is typically implemented with hardware locks). Thirdly, individual memory modules can only be accessed by one processor at a time, so all but one of the processors seeking access to a given module will be blocked for the duration of an individual processor's access (Ding & Densham 1996). And finally, these machines can not be scaled up infinitely. It can be difficult to optimise controlled memory access such that the application performance is scalable (increases linearly with the number of processors). As the number of processors trying to access memory increases, so do the odds that processors will be contending for such access. This quickly becomes a bottleneck to the speed of the computer (Ragsdale, 1991). A way round this is to introduce memory caches on each processor. However, this is the first step on the road to fully distributed memory machines!

3.2.2 Distributed Memory

In distributed memory configurations each processor has its own private memory, known as local memory. The processors are each connected to a small subset of the total number of processors and the method by which they communicate with each other is known as message passing. Message passing occurs when an operation on one processor requires data located in the private memory of another processor. Thus, an explicit message must be sent from one processor to another through an interconnecting network (see Figure 3-2).

The main consideration is how the processors will be connected and how they will communicate, as these have implications for both performance and efficiency. There are a number of options, but to connect all of the processors to all of the others is completely infeasible - for a large number of processors the number of connections required would comprise the square of the number of processors. It therefore makes sense to connect each processor to a subset of the total number of processors.

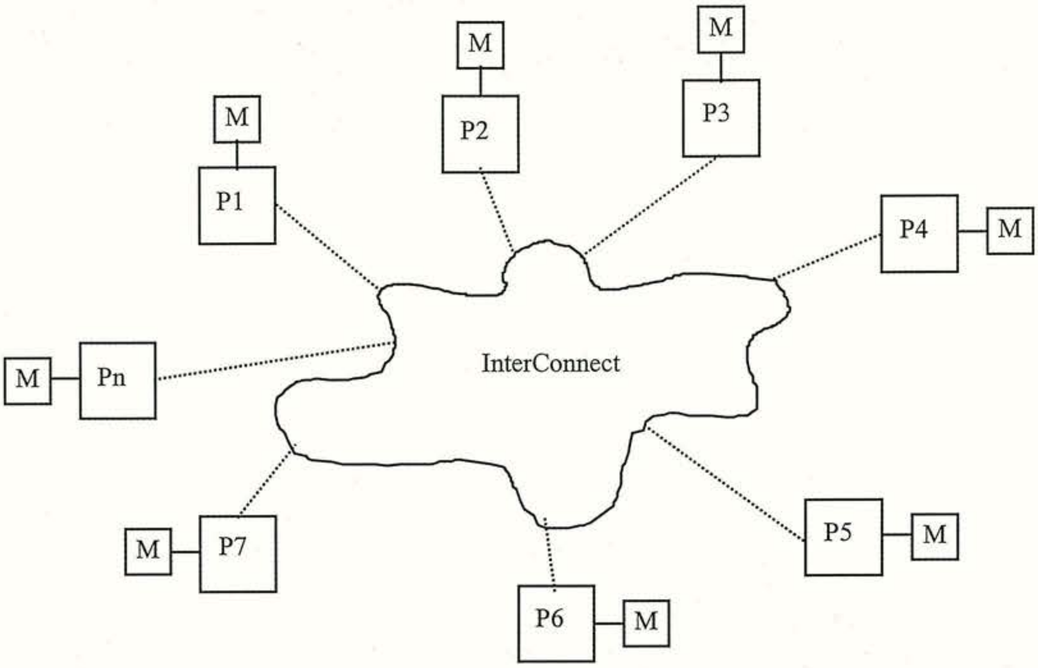


Figure 3-2:Schematic Architecture of a Distributed Memory MIMD Machine (after EPCC, 1992)

There are two options. Some computers, such as hypercube based machines, have a fixed topology of processors. Others use switching chips between processors which allow the user to adapt the technology to suit the particular program being run. Using four links per processor is fairly common practice, which allows the creation of many topologies such as processor trees, meshes and 3-D and 4-D hypercubes (see Figure 3-3). This approach is more efficient if local memory is accessed more frequently than other types of access because message exchange is proportional to message length (Denning & Tichy, 1990)

The need to perform explicit message passing is a feature of these machines that makes them harder to program and makes it difficult to produce a compiler. However, the crucial advantage is, if the application code is written to minimise the amount of message passing (and make message passing local whenever possible), then scalability is very good and large scale MIMD-DM machines offer a realistic route to Teraflop computing power, i.e. 10^{12} Floating Point Operations Per Second (FLOPS) (EPCC, 1992). By providing both uniform and fast memory access time, distributed memory systems offer higher absolute performance than shared memory systems for many scientific applications (Ragsdale, 1991). Designers of parallel algorithms favour distributed memory architectures as the majority of parallel algorithms can be designed to localise the data depending on each processor (Denning & Tighy, 1990).

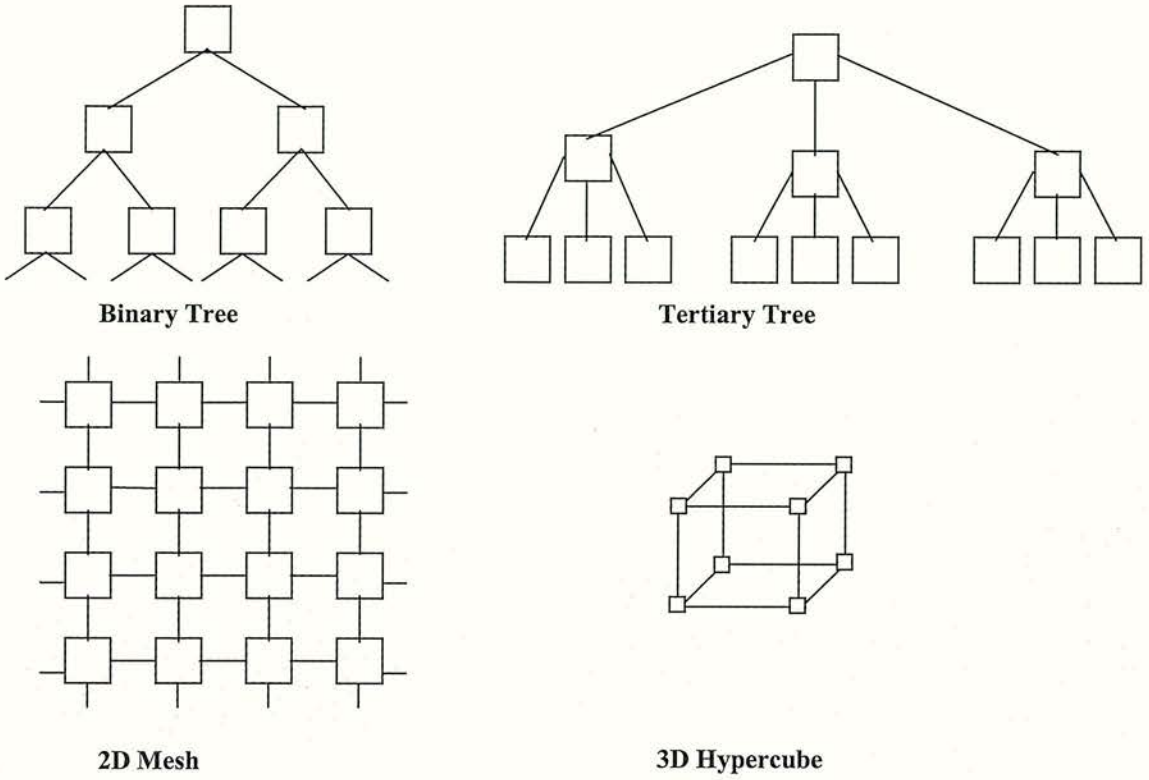


Figure 3-3: A variety of topologies with 4-link transputers (after EPCC, 1992)

3.2.3 Virtual Shared Memory

Virtual shared memory is a concept that has been introduced to try to bridge the programming gap with MIMD-DM computers. In the past, programmers had to specify the memory configuration in the program or at best write the program for the given memory configuration. Virtual shared memory frees the program from these constraints. Advances in technology mean that the distinctions between shared memory and distributed memory are becoming blurred. The processors in this architecture have access to the whole memory of the computer but, due to the hierarchical organisation of the memory, different parts can be accessed at different speeds. This type of access is known as Non-Uniform Memory Access (NUMA). There are a number of advantages to this - the ease of programming of shared memory is retained, without the hardware memory contention problems which ultimately affect the scalability of the machine. Programs can also be run which require more memory than is physically available on a single processor, allowing memory resources to be used more efficiently in a multi-user environment (Sawyer, 1998).

The programmer can address global memory space as in the MIMD-SM model, although the underlying hardware is MIMD-DM and retains the scalability of MIMD-DM architecture. How closely the programming code reflects the MIMD-DM model becomes a performance issue rather than one of feasibility.

3.2.4 Shared vs. Distributed Memory

All three of the architectures described have been implemented. Manufacturers of shared memory machines - often known as symmetric multi-processors (SMPs) tend to use smaller numbers of more powerful processors. SMPs are made by many leading manufacturers, e.g. Digital, Silicon Graphics, Bull, Sun Micro Systems and Intergraph. Many vendors of vector machines have also adopted them. SMPs have made a significant impact on the high performance computing market used as either parallel machines or servers achieving high throughput for sequential jobs. Existing software can be run and there is a low cost entry into the market, making them more attractive (Sawyer, 1998).

Distributed memory machines with thousands of processors have also been built. However, distributed memory requires more sophisticated configuration software and networking hardware and this can make small systems seem very unattractive. Clusters of work stations connected via a network are a distributed memory system - distinctions are made between distributed machines which are thought of as 'tightly coupled' and a collection of work stations which are known as 'loosely coupled' (Sawyer, 1998).

Arguments against shared memory machines scaling effectively with large numbers of processors are becoming less of an issue due to the advances in memory, bus and caching. Also many real applications will not themselves scale to large numbers of processors because they contain elements that are inherently sequential and there comes a point where nothing will be gained by adding further processors (Sawyer, 1998; Sloan, 1998)

3.3 Parallel Performance

The performance of parallel programs is a complicated issue because there are many factors involved. The main difference between parallel programming and the more conventional serial programming is that in parallel programming there are several operations that can occur at the same time on different data. Therefore, careful consideration should be given when designing programs to ensure that conflicts for data and resources do not occur. Some of the complex, inter-related areas for consideration are: data decomposition techniques used; the communications infra structure between processes and the mapping of processes to processors; the scheduling of tasks; and the balancing of processor work loads to ensure minimal overheads and maximum performance. The goal of parallel processing is to produce a program that balances all of these in such a way to optimise performance.

Parallel performance can be measured using several different criteria. These are:

- **program latency** - the time taken to execute the program.
- **bandwidth** - i.e. the throughput of repeated similar tasks.
- **speed-up** - the ratio of performance of the parallel program to that of the sequential program.
- **efficiency** - the ratio of speed-up to the degree of parallelism.

To realise the full potential of using parallel processing with geographic data there are a number of factors which must be identified such as, how to measure performance of parallel processing, how to identify those factors affecting the performance, and lastly how to achieve high performance (Ding & Densham, 1996).

The following subsections describe methods of measuring performance of speed-up and efficiency.

3.3.1 Speed-up

The measure of success of parallelisation is often measured in speed-up. This can be obtained by analysing the proportion of the program that can be run in parallel, and that which must be run sequentially. Amdahl (1967) discussed the effect of the sequential component of a process and the expression for the maximum speed-up is known as Amdahl's Law:

$$S(N) = \frac{T_s + T_p}{T_s + T_p / N}$$

Where S_N is the speed-up with N processors, T_S is the fraction of operations which must be performed sequentially, and T_P is the fraction of operations which can be performed in parallel. The expression shows that speed-up is determined by the sequential components of the task if the number of processors available is infinite (Sawyer, 1998).

Speed-up can also be defined as the ratio of the time required to complete a given task when implemented on a single processor and when implemented on a parallel computer using N-processors

$$S(N) = \frac{T_{sequential}}{T_{parallel}(N)}$$

Where *S(N)* is the speed-up with N processors, and *T(N)* denotes the time elapsed on a parallel computer with N processors. The sequential time $T_{SEQUENTIAL}$ can be described as

$$T_{sequential} = T(l)$$

This definition of speed-up refers to parallel tasks with a small grain size as real tasks could not be fully implemented in the memory of a single processor, particularly those used in distributed processing (Fox *et al.*, 1988). This suggests an important advantage - the ability of distributed memory computers to handle much larger problems than serial computers (Ding & Densham, 1996).

3.3.1.1 Scalability

Scalability is the way in which the speed-up of a program increases with the number of processors. A task is said to have good scalability if there is a roughly linear relationship between the speed-up obtained and the number of processors used, up to a large number of processors. The difficulty is in defining just what is thought to be a large number of processors. One rule of thumb is to assume good scalability if near-linear speed-up can be maintained up to the largest machine on which the task can be run. Amdahl's Law (1967) shows that this can only be achieved if the sequential part of the problem is small in comparison with that which can be performed in parallel (Sawyer, 1998).

3.3.2 Efficiency

The *efficiency* of parallel processing is measured as the ratio of speed-up to the degree of parallelism, and is bounded by 0 and 1. Linear speed-up gives an efficiency of 1:

$$\varepsilon = \frac{S(N)}{N}$$

where efficiency ε of speed-up *S* using *N* processors.

3.4 Parallel Program Decomposition

To turn sequential code into a parallel program the code must be broken up into its component parts, some of which are inherently sequential and some potentially parallel. Decomposing the potentially parallel parts such that a number of processors can work concurrently on the problem should result in a decrease of the time the program takes to run. However, decomposition frequently incurs an overhead usually in the message passing between processors.

Decomposition can be classified according to whether the problem is divided on the basis of function or data. This can be further sub-divided according to the extent of the data dependencies[2]. Three important decomposition techniques are described below - trivial, functional and data decomposition.

[2] There is some ongoing discussion over the nomenclature of decomposition methods especially those of data decomposition techniques.

3.4.1 Trivial Decomposition

This is the simplest technique and does not really involve decomposition at all. A sequential program that has to be run independently on lots of different inputs clearly has some parallel potential simply by doing a number of the sequential runs in parallel. Since there are no data dependencies between the different runs the limit to the number of processors used is the number of runs required to complete the task. The execution time of the runs will be the execution time of the most time consuming run in the set. Trivial decomposition can be exploited to provide almost linear speed-up if runs take a similar length of time.

3.4.2 Functional Decomposition

Functional decomposition is the first true decomposition technique, breaking the program up into a number of sub-programs. The simplest form of functional decomposition is the pipeline where input passes through each sub-program in a given order. Parallelism is introduced by having several inputs moving through the pipeline simultaneously. For example, the first element of input enters sub-program**1** and is processed. This element moves on to sub-program**2** to be further processed. As input element one moves to the second sub-program a second input element enters sub-program**1** is processed and then moves on to sub-program**2** as the first input element moves onto sub-program**3** etc. Parallelism in a pipeline is limited by the number of stages in the pipeline. For greatest efficiency all stages of the pipeline should be kept busy - this requires that all stages of the pipeline take the same length of time. The pipeline is then balanced.

Functional decomposition tends to be very problem oriented and the amount of parallelism is dependent on the program. This means that as the size of the input data set grows it may not be possible to exploit further parallelism. Thus, items of data in a large data set are not likely to be processed any faster than those in a small data set. Further, large datasets will take proportionately longer to be processed.

3.4.3 Data Decomposition

Data decomposition, as its name suggests, requires that the problem of parallelism is tackled by splitting a data set up over a number of processors, rather than decomposing a program.

Many problems involve applying similar or identical operations to different parts of a large data set and are ideal for data decomposition. Data decomposition techniques fall into two classes: those appropriate for predictably balanced problems (Geometric Decomposition) and those appropriate for unbalanced problems (Scattered Spatial Decomposition and Task Farming).

If there are no data dependencies in the data set, i.e. the result of an operation on a single data item can be computed without knowledge of the rest of the data set, then the limiting factor for parallelism is the number of processors available. However, the results of a computation usually require reference to other data points - most commonly neighbouring points.

Typical data decomposed MIMD-DM applications make use of a master and slave arrangement of processors. Here there are two modules of code - the master and slave modules. There is a singe processor running the master code, but many processes running the slave code. The master typically handles all I/O, including the user interface and the file system. Data are broadcast from the master to the slaves, each of which performs work upon its domain. The results are then gathered together during or after the run.

3.5 Load Balancing Techniques

Load balancing is the art of breaking a problem down into a series of smaller parts and distributing them over a number of processors such that all processors are kept busy and will all finish at roughly the same time. If the processes are distributed to individual processors then total execution time is dependant on the largest individual task (i.e. if a process is largely sequential then it will always be the lowest limit for execution - Amdahl's Law). However, execution time can be improved by better use of the other processors.

Unless the parallelism is trivial there is always the trade off between communication costs and ease of load balancing as the granularity of a decomposition is varied. Task-farms and other regularly-decomposed problems will automatically be well balanced for a sufficiently fine granularity, but other problems are not so simple.

Load balancing can be defined as:

load balance= mean(load)/max(load)

where the mean and maximum are taken over the set of processors in the computer.

3.5.1 Functional Decomposition

For functional decomposition load balancing is very much dependent on the nature of the problem. There are usually two solutions to the load balancing problem. Either the functional units of a problem are very carefully balanced, or load balancing is achieved by allocating a suitable number of processors to each functional unit and data are decomposed across the available processors.

3.5.2 Data Decomposition

There are two types of load balancing - *static* and *dynamic*. The former tries to achieve a good load balance solely by judicious mapping of domains to processes before it broadcasts them. Dynamic balancing involves the appropriate allocation, or re-allocation, of work to processes 'on-the-fly' and is in general a difficult technique.

3.6 Parallel Geo-Processing

Whilst the early work on parallel geo-processing focused on SIMD architectures, in the last few years the concentration has moved much more towards the use of MIMD computers (Ding & Densham, 1996). The use of parallel concepts and paradigms in the design of more efficient algorithms for processing geographic data is an important area of research. An overview of MIMD based research was compiled by Ding & Densham (1996), including parallel matrix methods for spatial analysis and modelling (Quinn, 1987), the development of parallel graph algorithms applied to vector based GIS and network analysis problems (Quinn & Deo, 1984), and image processing and real time image processing (including image algebra) (Preston & Uhr, 1982). More recently research has explored developing procedures for line simplification (Mower, 1996), interpolation (Armstrong & Marciano, 1996), and polygon line shading (Roche & Gittings, 1996). Many of the issues concerning the implementation of GIS algorithms have been explored (Healey & Desa, 1989; Armstrong & Densham, 1992; Hopkins *et al.*, 1992; Healey *et al.* 1998a) and are proving surmountable.

The recent developments in parallel relational database systems have brought parallel computers to the attention of business and research establishments alike. Relational database queries are suited to parallel processing because they consist of uniform operations applied to uniform streams of data (DeWitt & Gray, 1992).

The problem of searching a sorted sequence in parallel has attracted a good deal of attention since searching is an often performed and time-consuming operation in most database applications (Akl, 1989).

Prominent relational database vendors such as Oracle have been involved in a number of projects to port their software to various parallel platforms including Sequent, Meiko, Parsys etc. (Trew & Wilson, 1991).

3.7 Benefits of Parallel Computing for Corporate GIS

Potential users of parallel computers are attracted by a number of benefits:

- **Cost/Performance ratio**

  In cost terms the cost/performance ratio of parallel computers is very good value for money

- **Ultimate performance**

  The ability to have a much larger computing capacity in order to stay competitive or tackle new problems. With conventional computing systems the cost may be prohibitively expensive or just simply infeasible.

- **Scalable performance**

  Many large organisations would like to have access to technology capable of providing a scalable application solution that could be used flexibly throughout the organisation. This would allow the provision of large central facilities and smaller distributed installations all running the software application tuned to the cost/performance requirements of the site.

- **General purpose capability**

  The use of commodity components is seen by many user of MIMD systems as the basis for high performance systems capable of running common applications such as spread sheets, word processors and also databases. This provides an attractive business case for investing in the hardware.

- **Entry-level pricing**

  The entry level cost for parallel computing can be very low. An accelerator board with supporting software (e.g. compiler and parallel libraries) can be bought for a few thousand pounds.

The biggest development for parallel computing with corporate GIS is the move away from specialist components to commodity processors and disks. This has provided the impetus for research into areas that are likely to benefit corporate GIS customers, such as the parallelisation of common GIS functions, e.g. raster to vector conversion and polygon overlay. A project (Healey *et al.*, 1998a), undertaken by the Parallel Architectures Laboratory and the Edinburgh Parallel Computing Centre (EPCC) at the University of Edinburgh, was designed to devise parallel algorithms for these common GIS functions. It was sponsored by the Department of Trade and Industry and leading vendors selling parallel architectures, relational database management systems, and a number of GIS manufactures including ESRI, Smallworld and Laserscan,

These small steps towards parallel GIS are coupled with the promotion and uptake of commercially available parallel databases in many large organisations. It is only a matter of time before parallel technology is produced and marketed for more modest sized businesses, which require the speed of data handling developed for the larger sized companies, to support a smaller work load. If corporate GIS installations are to take advantage of these moves it is important to establish what requirements GIS has of the parallel architecture and database, and whether currently available GIS will be able to take advantage of the parallel environment. To be able to do that it is first necessary to consider what roles databases have with GIS and to consider what parallel databases can offer.

4. Database Management Systems for GIS

This chapter on database management systems for GIS has a twofold purpose. The first is to examine the development of database management systems for corporate GIS, and the second to introduce the main database components used in this research project. The investigation into the use of parallel database management systems required the development of two databases, a small pilot database and a large corporate database. The pilot database was designed for initial testing of the parallel database system and consisted of a number of tables containing artificial data. The corporate database was created for the second phase of testing, and due to the size and complexity of the data model and the data it contained, became a serious project in database design. Therefore, as well as considering database developments, the chapter also includes some detailed descriptions of the underlying structure of relational database management systems and database interface mechanisms. The structure is useful when understanding the design of the two performance test databases. The interface mechanisms are required to understand some of the design issues concerning two database tests harnesses that were created (see chapter ten).

The chapter is divided into three main sections. The first section charts the development of relational database systems and object-oriented database systems and their use with GIS. The main focus of this section is the relational database system (RDBMS), due to the extensive use of this type of database in both GIS and corporate IT. The second section describes two different approaches to GIS: the hybrid and integrated models, and their associated data storage mechanisms. This includes a brief description of object-oriented (O-O) systems and the differences between the O-O approach and hybrid and integrated GIS approaches. The third section focuses on SQL, a relational database language. Several areas are explored: these include the appropriateness of SQL as an interfacing language between GIS and relational databases. The rest of the section describes the use of embedded SQL as an interfacing language between the database and external applications.

4.1 Introduction

The importance placed upon database management systems for corporate GIS and MIS has increased significantly as the need for accurate, timely, pan-organisational data has grown. The development of data integration strategies, e.g. data warehousing, and data analysis techniques are broadening data processing requirements of individual database management systems. There is a move away from concentrating simply on the performance of transaction processing loads from large financial systems and billing systems to supporting the much longer, complex, transactions belonging to exploratory GIS and MIS queries. The integration of corporate datasets for use with a variety of GIS and MIS applications will inevitably lead to a very mixed work load for databases. Work load mix is known to lead to difficulties in maintaining the performance of applications accessing a single database, and it is set to worsen given the increasing size of data sets available, for example legacy systems, satellite imagery, and the vast number of database hits generated by applications accessible through the internet.

As corporate GIS requirements grow, a need for the processing power of parallel database management systems is emerging. Relational database management systems, used by many businesses, are very well suited to parallelisation (DeWitt & Hawthorn, 1981; Bitton *et al.*, 1983) and a number of commercially available parallel relational database systems and parallel database engines are available. The move towards more integrated corporate GIS, where much of the data would be stored externally, opens the door to the use of parallel databases and much needed data processing power.

It must however be borne in mind that greatly increased data processing capabilities alone can not provide the total solution. The GIS models, where data are held outwith the application in external databases, are heavily reliant on the efficiency of their interfaces between the GIS software and database management system. A clumsy or inefficient interface will produce very poor data retrieval performance and form a major bottleneck for any GIS function requiring data from disk. Performance of GIS/database interfaces will inevitable become even more critical with the use of larger data sets and increased numbers of users.

Approaches to corporate GIS are also changing. The requirement to access data from many different sources has increased demand for efficient database interfaces to many of the most popular, general purpose, data management systems. Conversely, as MIS has become more sophisticated, and IT strategies deliver more integrated information systems, there is a growing requirement to access data held in GIS. There is now a demand developing for both GIS data and MIS data to be available in a common database system.

Prior to the widespread adoption of relational databases and entity-relationship data modelling approaches (Chen, 1976), a variety of special purpose file formats were used to store digitised cartographic datasets, although frequently the topological relationships between cartographic elements were not actually stored in conjunction with co-ordinate data. In the mid 1980s two approaches to storing GIS data were developed which made provision for data stored externally to GIS. They were known respectively as the *hybrid* model (Morehouse, 1985) and the *integrated* model (McLaren & Healey 1992). The hybrid model handled digital cartographic data in a proprietary file system for speed, while attribute data for map features were managed in a relational framework, for the convenience of the user. In contrast, the integrated model (as demonstrated by Van Roessel & Fosnight (1984)) was able to handle both co-ordinate and attribute data in a database environment external to the GIS (Healey & Waugh, 1987).

In the mid 1980s there was strong pragmatic justification for using the hybrid approach despite the additional software layers required to manage the linkages between the digital cartographic and associated attribute data. This was due to the relatively poor performance of relational systems at the time. Even a modest sized digital map coverage could involve millions of (x,y) co-ordinate pairs which would have resulted in extremely large database tables, whose rows could not be retrieved to the graphics screen for map display in any reasonable time (Bundock, 1987).

Although, the integrated GIS model fits more readily into the corporate IT data integration policies being developed and encourages the development of data exchange, hybrid approaches (e.g. ESRI Arc/Info) still dominate the commercial market place. However, even those strictly hybrid systems are moving towards a more integrated approach as the performance of relational database systems begin to outstrip their own in-house database systems, and demand for external database access has increased. For example ESRI (1999) has introduced a Spatial Database Engine (SDE) which provides an open interface between the user and all of the spatial data in an organisation, whether stored in a database management system or ESRI's native file structures.

4.2 Relational Systems

The drawing together of the concepts of digital cartography and database management has been a key element in the rapid expansion of corporate GIS, from very modest beginnings through to a major sector in the IT industry (Maguire *et al.*, 1991). The development of GIS in the 1980s coincided with the rapid adoption of relational database technology (Date, 1986), and thus the adoption of relational methods at an early stage of GIS development. They continue to be a dominant force for database management in GIS (Healey, 1991) despite the growth of interest in the use of object-oriented data structures (Worboys, 1992) and the availability of systems using object technology, such as Smallworld and Laserscan Gothic (Chance *et al.*, 1990).

The relational data model was first proposed by Codd (1970) at a time when both hierarchical and network models dominated the market place. A prototype relational database management system, System R (Astrahan *et al.*, 1976), was developed by IBM researchers and was designed to prove the practicality of the relational model by providing an implementation of its data structures and operations. The implementation proved an excellent source of information about concerns such as concurrency control, query optimisation, transaction management, data security, recovery techniques and user interfaces (Ricardo, 1990). The model has since been used by many other DBMSs in mainframe, mini, workstation, PC, and now parallel, environments, including applications such as Ingres, Oracle, DB2 and Access. The popularity of the relational model has encouraged vendors of non-relational systems to provide a relational user interface, regardless of the underlying model (Ricardo, 1990).

The relational model is based on the mathematical concepts of relations and sets which have been expanded to apply to database design. The power of mathematical abstraction and the expressiveness of mathematical notation has led to the development of a simple but powerful structure for databases. The simplicity of the model makes it easy to understand at an intuitive level. It allows a separation of logical and physical considerations such that the logical design can be performed without concerns about storage. Logical data notions can be expressed in a manner that are easily understood. Data operations are also easy to express and do not require users to be familiar with the storage structures used. Finally, the model uses a few powerful commands to accomplish data manipulations that range from simple to complex (Ricardo, 1990). These are the reasons it has become so popular and also why it has been adopted by a large number of GIS.

4.2.1 Relational Database Design

4.2.1.1 Relational Data Structures

The relational model is based on the concept of a relation (or set), that is physically represented as a table. Tables are used to hold information about the objects to be represented in the database and represent the relationships among all the attributes contained in the table. Each row in the table (termed a tuple) represents a fact - a permanently related set of values. Each column represents an attribute, or a single value.

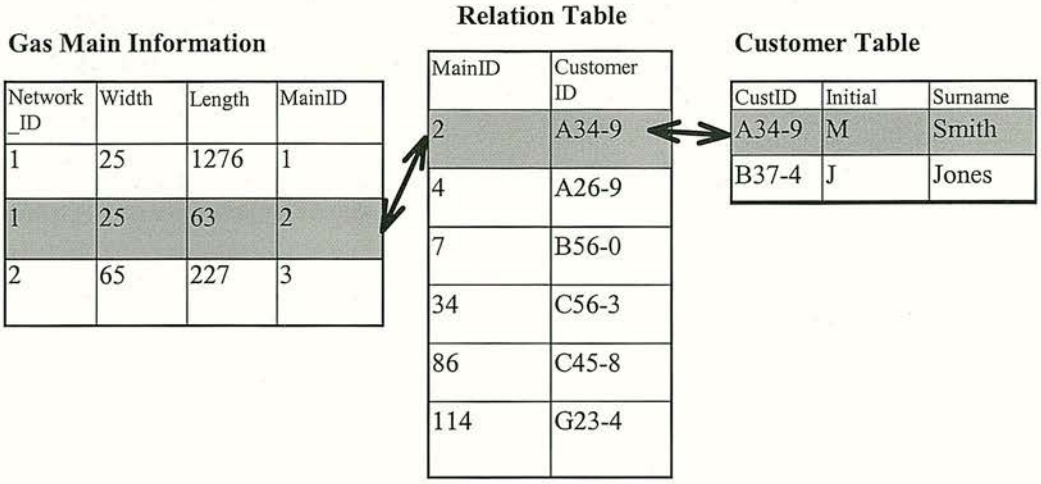
The characteristics of tables result from the properties of relations. As a relation is a set, the characteristics of the two are similar:

- the order of tuples is immaterial
- there are no duplicate rows
- each element of the tuple is atomic, i.e., contains a single value.

Where the mathematical relation and the table in a relational model differ is that the order of elements in the tuple of a mathematical relation are important: (1,2) is very different from (2,1). In the relational model the order of elements is of little importance because each column has a heading identifying which attribute the value belongs to. The primary key of the table is an attribute, or group of attributes, that describes each tuple in the table uniquely, and so can be used to identify the tuple (Ricardo, 1990).

4.2.1.2 Relational Joins

The mechanism for linking data in different tables is called a relational join. Values in a column or columns in one table are matched to corresponding values in a column or columns in a second table. From the second table a further match to a third table can be made and so on until the necessary data from the requisite number of tables have been retrieved (see Figure 4-1).



Gas Main Information

Network _ID	Width	Length	MainID
1	25	1276	1
1	25	63	2
2	65	227	3

Relation Table

MainID	Customer ID
2	A34-9
4	A26-9
7	B56-0
34	C56-3
86	C45-8
114	G23-4

Customer Table

CustID	Initial	Surname
A34-9	M	Smith
B37-4	J	Jones

Figure 4-1: Relational Tables with a Relational Join - simplified example from the Gas Main database

One-to-one, one-to-many and many-to-many relationships can be represented in a relational database. Unlike other types of database, relationship sets describing many-to-many relationships between entity sets, are also represented by a table of data values. The tables contain columns which reference the entity sets being related, together with further columns for any attributes of the relationship itself. Since relationships between entities are directly represented as tables, there is no requirement for pointers or linkages between data records to be set up.

4.2.1.3 Advantages of relational systems

The advantages can be summarised as follows:

- a rigorous design methodology based on sound theoretical foundations;
- all the other database structures can be reduced to a set of relational tables, so they are the most general form of data;
- ease of use and implementation of applications compared to other types of system;
- modifiability, which allows new tables and new rows of data within tables to be added without difficulty;
- flexibility in *ad hoc* data retrieval because of the relational join mechanism and powerful query language facilities (Healey, 1991).

The important advantages of the relational approach and the availability of good proprietary software systems such as Oracle, Ingres and DB2 have contributed greatly to the rapid adoption of this technology, both in the GIS field and in automated data processing operations of all other kinds, since the beginning of the 1980s. The relational model has the flexibility to link GIS models to corporate databases, and the extensibility to provide a base for other database models such as Object-Oriented systems. Relational systems now dominate the market for DBMS in the GIS sector and this is expected to continue for the foreseeable future.

4.2.1.4 Disadvantages of Relational Systems

Many of the disadvantages identified in relational systems were in comparison with the performance of hierarchical and network systems, which for the most part have been dropped in favour of relational database management systems. For example, the manipulation of data in relational tables, based on matching values, was a much more time consuming operation than using physical pointers or links as used in hierarchical and network models (Aronoff, 1989). Also, they could be more difficult to implement, and were, initially, slower in performance than the other two models. However, speed has become much less of an issue as processors become more powerful and methods for data retrieval and indexing become more efficient. The advent of parallel database management systems has not only improved processing speeds but has also greatly improved the data handling capacity, making it possible to store and manipulate terabytes of data.

4.3 Object-Oriented DBMS

The most recent developments in GIS design have taken their lead from object-oriented methods. Object-oriented models are based on objects which must be:

- Identifiable;
- relevant (be of interest);
- describable (have characteristics) (Mattos *et al.*, 1993).

The concept of *object* is central to the object-oriented approach. An object can be defined, in its basic form, as an entity that has a static data aspect and dynamic behaviour. The static part is represented by the values of local variables (known as instance variables). The combined attributes of the object constitute its *state.* The dynamic behaviour is expressed as a set of operations or methods (known as instance methods) that operate on the object under certain conditions (Somerville, 1989; Rowe, 1986; Worboys, 1994).

Individual objects belong to a class, which defines the type of object. Each class has a superclass from which it can inherit both instance variables and methods. For example, an object class called *polygon* may be defined which is also the superclass for another class called *land parcel*. All the instance methods and variables of the polygon superclass are inherited by the land parcel class, unless they are re-defined at the land parcel level.

4.3.1.1 Object -Oriented Databases

An object-oriented database management system (OODBMS), in addition to supporting the object oriented constructs described in the previous section, must provide an environment for the management of objects and their hierarchies, ideally providing all the features of modern databases in the OODBMS:

- schema management, including the ability to create and change class schemata
- a usable query environment, including automatic query optimisation and a declarative query language
- storage and access management
- transaction management, including control of concurrent access, data integrity and system security (Worboys, 1994).

However, there are some technical problems in implementing some of these facilities. Query optimisation is made difficult due to the complexity of the object types in the system. There may be a multitude of methods implementing operations and for each of which there may be no measure of the implementation cost (Worboys, 1994). Another problem is that of indexing (Worboys, 1994). Relational databases rely on direct access to attributes, while objects are accessed by messages and identified by their object identifier.

There are a number of commercially available object-oriented databases in use for many different platforms. POSTGRES is probably the most well known, with a wide user base. It has had a rather ad hoc evolution and is currently bundled with LINUX – PC Unix freeware. Two other available systems are JADE (1999), running on WindowsNT and RS6000 and Orbit by Bridger Systems (1999). There are at least two mainstream object-oriented GIS environments: Smallworld, and Gothic from Laserscan.

4.4 Hybrid and Integrated Approaches to GIS Database Management

An important distinction in the design of GIS is between systems in which the standard file system is used to store the digital map co-ordinate data with attribute data held in a linked DBMS (the so-called hybrid approach), and those in which both types of data are to be held in the DBMS (the integrated approach). Use of the computer file system directly, rather than through the intermediate step of the DBMS, will generally yield faster response times. On the other hand, the DBMS provides a wide range of ready made data manipulation tools so that programming effort can be concentrated on algorithms for spatial analysis, user-interface requirements and data distribution.

4.4.1 The Hybrid Data Model

The starting point for this approach is the view that data storage mechanisms that are optimal for locational information are not optimal for attribute/thematic information (Morehouse, 1985; Aronson, 1985). On this basis, digital cartographic data are stored in a set of directly accessed operating system files for speed of I/O, while attribute data are usually stored in a standard commercial relational type DBMS such as INFO, Oracle, INGRES or INFORMIX. The GIS software manages linkages between the cartographic files and the DBMS during different map processing operations such as overlay. While a number of different approaches to the storage of the cartographic data are used, the linking mechanism to the database is essentially the same. It is based on unique identifiers stored in a database table of attributes that allow them to be tied to individual map elements.

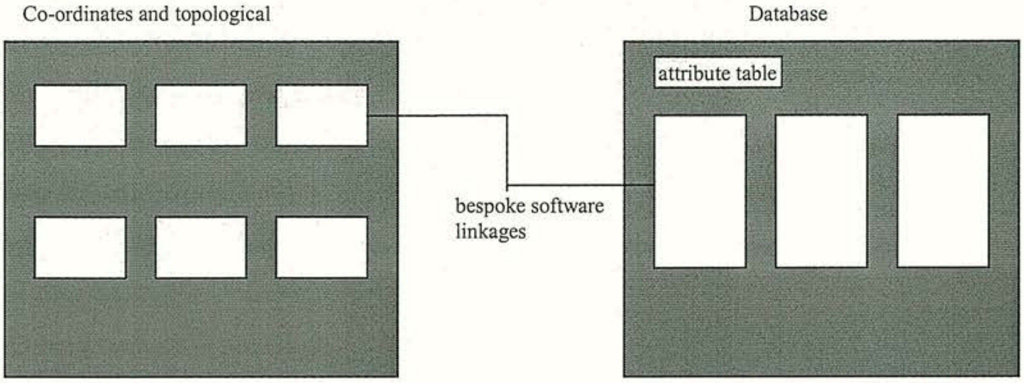


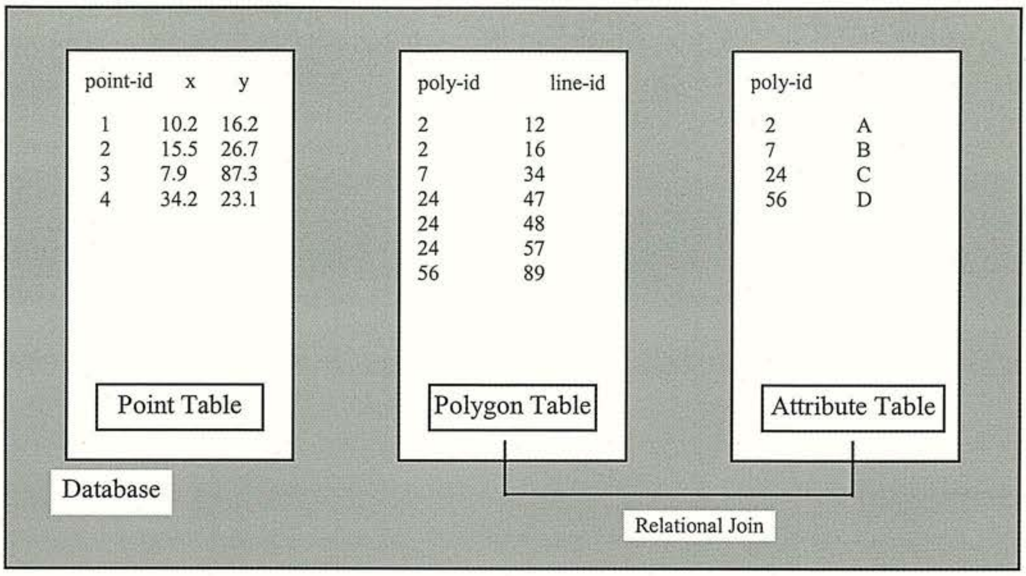
Figure 4-2: The Hybrid GIS Model (after Healey *et al.*, 1991)

4.4.2 The Integrated Data Model

The integrated data model approach is also described as the spatial database management system approach, with the GIS serving as the query processor sitting on top of the database itself (Guptill, 1987; Morehouse, 1989). Most implementations to date are of the vector topological type, with relational tables holding map co-ordinate data for points/nodes and line segments, together with other tables containing topological information, in a manner partly similar to that described by van Roessel (1987). Attributes may be stored in the same tables as the map feature database or in separate tables which can be accessed using relational joins.

Van Roessel's work provides a detailed analysis of how digital cartographic data can be represented using correct relational database design methods. However, in practice this has proved to be unsatisfactory in implementation because of performance overheads. This is particularly so when *(x,y)* co-ordinate pairs for individual vertices along line segments are stored as different rows in a database table. To achieve satisfactory retrieval performance it has been found necessary to store co-ordinate strings in *bulk data* or *blob* (binary language object) columns in tables (Healey, 1991).

The performance issue must also be addressed in respect of very large digital cartographic databanks since both the U.S. Geological Survey (Guptill 1986) and the Ordnance Survey (Smith, 1987) have developed integrated data models for national mapping applications. Both these applications are suitable for implementation in a relational database framework (Healey & Waugh, 1987).



point-id	x	y
1	10.2	16.2
2	15.5	26.7
3	7.9	87.3
4	34.2	23.1

poly-id	line-id
2	12
2	16
7	34
24	47
24	48
24	57
56	89

poly-id	
2	A
7	B
24	C
56	D

Figure 4-3: The Integrated GIS Model (after Healey *et al.*, 1991)

4.4.3 Object-Oriented Model vs. Hybrid & Integrated Model

One of the major differences between an object-oriented approach and that of the hybrid or integrated approaches, is that not only are the attributes (geometric or otherwise) of entities or objects stored within the GIS/database, but so also are the allowable operations that can be performed on them. From the database viewpoint this increases the complexity of the implementation. This is, however, regarded as justifiable because of the benefits of encapsulation of object attributes and behaviour for overall system design and ease of customising (Smallworld, 1991b). Systems of this type may also provide interfaces to external relational or other types of DBMS, in addition to their own internal object managers.

4.5 Structured Query Language (SQL)

The move towards a more integrated corporate GIS structure has begun to turn attention away from issues involving pure database performance towards database interfacing issues (described above). One of the most obvious interface mechanisms for relational database management systems is SQL, the structured query language developed to interrogate relational database systems. The following sections consider what qualities SQL has to provide a successful interfacing mechanism, and where the weaknesses lie. Included in the discussion are a description of the basic elements of the SQL language, and embedded SQL, which consist of SQL statements wrapped up in another programming language such as C.

SQL was designed as a high level interface to relational database systems to manipulate tables and has been successfully used as a database interface for applications that can easily be expressed in terms of tables (Egenhofer, 1992). It was the language adopted for IBM's System R project, a research prototype in the mid-1970s (Astrahan *et al.*, 1976) and was first introduced as a commercially available implementation by the ORACLE Corporation in 1979. SQL has also been implemented in IBM's DB2 and SQL/DS database systems and many others.

The American National Standards Institute (ANSI) adopted SQL as a standard language for relational database management systems (RDBMS) in October 1986 (ANSI X3.135-1986). The SQL standard has also been adopted by the International Standards Organisation (ISO standard 9075) as well as the U.S. Federal Government, in the Federal Information Processing Standard (FIPS) number 127 (Oracle Corporation, 1990).

The latest SQL standard, often know as SQL-92, has been published by ANSI and ISO (ANSI X3.135-1992), "Database Language SQL"; ISO/IEC 9075:1992, "Database Language SQL") (Oracle, 1997c). It has three levels of compliance, Entry, Intermediate and Full. The new standard adds new features and capabilities to the specifications. It contains a better definition of direct invocation of the SQL language, improved diagnostic capabilities, a new status parameter (SQLSTATE), a diagnostics area, and supporting statements (SQL92_STANDARD).

Most recently, SQL3 is under discussion at the moment. The options under consideration at the moment are to either specifically define spatial extensions for GIS within the language, or provide the mechanisms for user groups such as GIS to make their own extensions.

SQL is described as a structured query, update, data definition, control, consistency, concurrency and dictionary language by the ORACLE Institute (Oracle Corporation, 1985a). It is a non-procedural language that processes sets of records rather than just single records, and it provides automatic navigation of the data. SQL allows the user to work with higher level data structures by managing sets of records. The most common form of a set of records is a table. All SQL statements accept sets as input and they all return sets as output. This set property allows the results of one SQL statement to be used as the input to another SQL statement.

SQL provides statements for a number of tasks including:

- querying data
- inserting, updating, and deleting rows in a table
- creating, modifying, and deleting database objects
- controlling access to the database and database objects
- guaranteeing database consistency

Previous database management systems often used a separate language for each of the above categories.

4.5.1 SQL in GIS

The SQL language has been used by many groups of users for many different tasks. These range from: a high-level interface language for GIS to manipulate data stored in external databases; a data exchange language to transfer data from one database to another; to an *ad hoc* database query language. The language requirements for each of these roles is very different: an interface between a database and an application must fit into a high level programming language; while a query language must consider the interactions between humans and the computer (Egenhofer, 1992) The ability to encompass all these different requirements in one universal language would be almost impossible.

In an effort to circumvent some of the difficulties presented by SQL as a non-procedural language, a number of SQL variants have been created. Oracle provide SQL*PLUS to query the database, PL/SQL - SQL with procedural extensions, and embedded SQL to be incorporated into programs external to the database (Oracle Corporation, 1990). There are also groups working on a standard for GIS extensions in SQL. The development of a standard for GIS extensions in SQL would greatly enhance the exchange of information for both hybrid and integrated corporate GIS. It would have an effect on both data retrieval, greatly reducing the amount of code required for functions not yet supported (e.g. adjacency tests) and performance of spatial SQL, retrieval methods could be optimised for individual functions. It should also improve the performance of GIS/database interfaces. As common interchange and data access methods are developed it should greatly reduce the work load generated by interface queries by devising standard procedures for accessing data, and eliminate much redundant work, such as submitting the query for every row of data returned. Standards, once developed, also tend to concentrate research efforts on improving them; thus the performance of such interfaces should improve.

SQL has become the standard for relational database management systems. Designed as a high level interface to RDBMS it has successfully been used as a database interface for applications that can easily be expressed in terms of tables. For *non-standard* applications such as GIS, it has so far proved to be a restrictive model and insufficient for solving some typical GIS tasks (Egenhofer, 1992). The main ingredient lacking is a support for spatial operations such a adjacency, difficulty in incorporating concepts such as graphical display and specification, and a lack of power and support for qualitative answers, knowledge queries and metadata queries (Egenhofer, 1992).

There have been a number of attempts to overcome some of the short comings of SQL and numerous extensions have been proposed, for example, proposals to deal with complex objects (Lorie & Schek, 1988; Mitschang, 1989), temporal data (Date, 1988; Ariav, 1986; Sarda, 1990) and spatial data (Roussopoulos *et al.*, 1988; Ooi *et al.*, 1989, Egenhofer, 1991; Raper & Bundock. 1991). There have also been some limited extensions such as locate and join made available in commercial database software (Oracle Corporation, 1997d).

However, for all its limitations as a spatial query language, it will continue to be used as the major database interface language between GIS and RDBMS because it has a proven track record in reliability, and is unlikely to be replaced until new reliable methods have been developed.

4.5.2 SQL Implementation

At its most basic, SQL is an implementation of the five relational operations - selection, projection, Cartesian product, set union and set difference (Codd, 1970) - as well as a few other useful operations such as aggregation operations and views. The syntax is based on the SELECT-FROM-WHERE clause framework, corresponding to the relational operations of projection, Cartesian product and selection respectively (Egenhofer, 1992).

```
SELECT <target list>
     FROM <list of relations>\
     WHERE <condition> (Oxborrow, 1986)
```

For example, given two relations **MAIN** and **VALVE** with the attributes **main_id**, **main_diameter** and **valve_id** and **main_id**, the following query will return the valve identifiers of all valves situated on mains whose diameter is greater than 6 inches

```
SELECT valve_id
     FROM main, valve
     WHERE main_diameter > 6 and
           main.main_id=valve.main_id
```

The result is a relation displayed as a table.

4.5.2.1 Relational Joins

A relational join is a form of SELECT statement that combines rows from two or more tables. Each row of the result will have column data from more than one table. The join occurs whenever multiple tables are referenced in the FROM clause of the SELECT statement. The optional WHERE clause determines how the rows of the table are combined.

A simple join, the most common type of join, returns rows from two tables based on an equality condition in the WHERE clause *e.g.*

```
WHERE <table1.columnname> = <table2.columnname>
```

This is also known as an *equi-join* because it uses an = comparison operator in the WHERE clause.

SQL is able to handle null values when a *not known* situation arises. The *null value* in SQL is a special value used to indicate the absence of any data value (SQL92_STANDARD).

SQL allows the user to create a database without having to define all of the tables at the beginning. This makes it possible to create and drop tables, or alter existing tables at any time during the implementation phase.

4.5.3 Embedded SQL

Embedded SQL refers to the use of standard SQL statements embedded within a procedural programming language such as C, FORTRAN, Pascal or Ada, providing a mechanism for database integration from outwith the database environment. It consists of a collection of SQL statements such as SELECT and INSERT, and flow control commands that integrate standard SQL commands within a procedural programming language.

Embedded SQL is used extensively in this research for a number of reasons. The first is for the purpose of collecting performance statistics. Performance statistics were gathered from Oracle about the processing and management of SQL queries submitted to the database from external sources, by querying systems tables. The performance gathering queries formed part of a much larger test harness (see chapter ten), written in C, that collected a whole range of data about the response of the parallel database environment to different types of query and work load, submitted using SQL as an interface mechanism. The second reason for using embedded SQL was to allow SQL queries to be submitted at specified times of the day using Unix batch queues. This was particularly useful for those tests timed for 2 or 3am. The final reason was to create a multi-user workload rapidly for parallel database environment. By submitting embedded SQL queries using PRO*C, the Unix system could control and generate both users and workload.

Embedded SQL is interpreted by a number of precompilers which translate the embedded statements into commands that can be understood by the procedural language compilers. Pre-compilers exist for Ada, C, COBOL, FORTRAN, Pascal and PL/I (SQL92_STANDARD). The programming language used for the research project was C and the precompiler was PRO*C.

The following subsections briefly describe some of the elements involved in embedded SQL.

4.5.3.1 Host Variables

A *variable* within SQL is used as a place holder for a value within an SQL statement. These variables function in a similar manner to programming language variables. Before a variable can be used it must be *bound* to a host language variable in order that the value of the host language variable can be accessed by the database. Host variables were used in the research project to pass information to the database as part of the logon procedure, and were used to update various tables.

There are two types of statement in which host variables can be used. The first is where the nature of the transaction can be predetermined to a great extent. These are used in conjunction with the keyword INTO. For example, extracting data stored in certain fields into host variables so that the data can be further manipulated by the host programming language.

4.5.3.2 Cursors

A cursor is a work area used both by the database and the programming language interface to store the results of a particular query. A single cursor is associated with a single SELECT statement, and may be executed repeatedly for different variations of the query (using different host variables). Using a cursor requires the following basic commands:

- DECLARE CURSOR
- OPEN CURSOR
- FETCH
- CLOSE CURSOR

Once a cursor is opened, it is used to retrieve multiple rows from its associated query. All the rows which satisfy the criteria of the query form a set, called the *active set* of the cursor. Rows of the active set are returned, one by one, by fetching them. Then the cursor is closed.

There were a number of reasons for exploring the mechanisms of data retrieval. The first was to decide how to construct the performance tests e.g. what to test, how to test it and how to interpret the results. By analysing the different stages of data retrieval using embedded SQL it is possible to isolate those areas where there was high resource usage, where there was a potential for possible bottlenecks. Also important was the design of the performance tests to ensure that extra stages, inserted in the test harness, to aid in the gathering of statistics did not adversely affect the overall testing procedure by creating a high level of resource usage or create bottlenecks.

4.5.3.3 Declare Cursor

DECLARE CURSOR defines a cursor by assigning it a name and associating it with a query. This is a declarative SQL statement and must occur before any statement referencing the cursor.

4.5.3.4 Open Cursor

By opening the cursor the query is evaluated and the *active set* of rows identified. It is here that the input variables of the WHERE clause (if any) are evaluated, identifying which rows satisfy the query. The cursor is placed into an *open state* and is placed just before the first row of the active set.

Once the cursor has been opened the input variables are not re-examined until the cursor is re-opened.

4.5.3.5 Fetch

This reads the rows of the active set and names the output variables that will contain the results.

The first time FETCH is executed the cursor moves from above the first row to the first row. This row then becomes the current row. Each successive FETCH advances the cursor by one row. The cursor may only move forwards through the active set. To go back to an earlier row it is necessary to close and re-open the cursor.

4.5.3.6 Close Cursor

Once a cursor is closed it releases the resources it was using. No FETCHes may be executed against a closed cursor as the active set becomes undefined.

Although cursors can be explicitly declared for PRO*C operations, PRO*C will automatically create cursors for other operations. The following types of cursor exist:

- Explicitly declared cursors - These are logical cursors declared within a program with cursor commands and are used to fetch query results.
- Implicitly opened cursors - Logical cursors that PRO*C has obtained on behalf of the user, but requiring no action by the user.
- Database cursors - Physical cursors which are required for everyday operations.

A procedural extension of SQL called PL/SQL also exists which allows the user to link several SQL commands through procedural logic. Although this extension is available it must be stressed that this is not currently part of any standard.

4.6 Summary

The design of GIS reflects the improvements in database management systems. The hybrid GIS model, developed at a time when the performance of relational database systems was relatively poor, has proved a popular and successful model. However, integrated GIS models, using commercially developed databases have begun to emerge as the performance and data-handling capabilities have dramatically improved. The rapid expansion in the use of corporate GIS and the increasing requirements from users for greater GIS functionality have encouraged vendors to use databse systems developed to manipulate huge volumes of data and concentrate on GIS capabilities.

Use of the integrated GIS model has placed much more demand on the interface mechanisms that exist between GIS applications and the database systems. Languages such as SQL for relational database systems provide a means of interfacing the GIS and database. Although SQL was not developed for this particular task, it has the advantage that GIS extensions to the language exist, to make interfacing more efficient. SQL can also be embedded into many other programming languages such as C, Fortran and Pascal, which provides a flexible means of developing interfaces. There are a number of shortcomings associated with using SQL as an interface between GIS and a relational database, but these are known and not insurmountable.

Relational database management systems are the most common type used with GIS, although O-O GIS environments such as Smallworld and Gothic are available. Relational database management systems have a long history of development and have honed their data retrieval, security and recovery methods. However, they are not the perfect solution to GIS database requirements. Object-oriented databases may well provide some of the solutions particularly where modelling power and performance are required, or large databases are being processed and older versions need to be kept (Noervaag, 1999). The performance of parallel databases, particularly for transaction processing is unrivalled and the development of parallel relational database management systems has provided the ability for massive data processing, particularly for data warehousing. The main problem is that they are optimised for read operations. As the main memory capacity increases, the number of write operations also increases, and therefore there is a need for write optimised databases.

Object-orientated databases may eventually provide a viable solution to the SQL/relational database shortcomings. Parallel object-oriented databases are in the research stage of development and are being considered for use specifically with software applications such as GIS, and may prove to be an excellent solution for some GIS applications, in particular modelling. However, until commercially available, and viable object-oriented databases become available, parallel relational databases are providing a useful solution to many of the data handling issues identified.

SECTION 2

Functionality and Performance Considerations for Parallel Databases

The use of commercially available parallel databases with GIS is in its infancy. The previous chapters in the background section have shown that there is a demand for corporate GIS and a requirement for hardware and software solutions to solve issues identified as potential problems in the near future, such as the need for faster processing and the capability to manipulate and analyse much larger datasets in something approaching real time. Parallel architecture has developed to a stage where it has become much more attractive to business because it is no-longer wholly dependent on specialist hardware and software. Symmetric multi-processing (SMP) has started to lead the way towards the corporate use of massively parallel machines, which can run software designed for both serial and parallel architectures. The IT revolution has provided roles for both corporate GIS and parallel servers which are being accepted by businesses and promoted by vendors such as SAS, Oracle and Microsoft.

Parallel databases have become a serious software option with a number of large vendors providing access to parallel database management software, for example, Meiko and nCube (Statler, 1993).

What are missing thus far are the benchmarking techniques for measuring and comparing performance of serial and parallel versions of the software, and standard benchmark statistics to categorise the performance of competing parallel database management software. However, before considering the developments of commercial benchmarks it has to be established whether an experimental parallel GIS environment, consisting of parallel architecture, standard GIS applications and a parallel database, will provide a solution to any or all of the technical deficiencies identified earlier and establish areas of technological and organisational risk. To test these points it is necessary to design a series of benchmarking performance tests to be able to measure performance of a range of functions and strategies. These tests include:

- interfacing GIS software with the parallel database for both serial and parallel architectures
- exploring different models for integrating GIS and databases
- examination of storage, manipulation and retrieval of digital cartographic data.

Therefore, there is need to consider what it would be appropriate to measure, which particular indicators should be used, and how those chosen indicators should be collected and analysed.

5. Parallel Database Management Systems for GIS

From the overview of database developments in Chapter four it is apparent that important opportunities for the use of parallel database technology exist in both the integrated and hybrid data model approaches. Systems based on the integrated approach offer a very flexible solution for data storage and retrieval, many steps ahead of the hardware database engines of the 1980s. The hybrid approach, offers the potential to meet both transaction processing and *ad hoc* query requirements for centralised MIS systems (Healey *et al.*, 1998b).

There is a twofold purpose to this chapter in reviewing the development of parallel database technology. The first is in the context of GIS requirements, highlighting areas of benefit as well as the issues and difficulties that remain. The second, more practical purpose , is of identifying issues pertinent to the development of a suitable environment for exploring performance testing of a parallel database with corporate GIS applications. To be able to design and apply a suite of tests to the corporate GIS/parallel database configuration it is important to consider which aspects are of primary importance and how they are to be measured. This way the type, content and reporting mechanisms of individual performance tests can be defined.

There are two main areas of development in parallel database management systems - hardware and software, and consideration of these comprises the first two sections. The third considers the parallelisation of database queries, how they can be decomposed across a number of processors, and what effect query content has on parallelising the query. The final section describes some of the obstacles that still have to be tackled.

5.1 Hardware

The history of parallel database systems is full of proposals for different hardware solutions for improving the performance of database operations. They range from dedicated database machines with imaginative hardware solutions for rapidly searching for, and retrieving, data to contemporary parallel machines which consist of commodity components relying on the software to provide parallel performance.

5.1.1 Database Machines

Hardware database machines represent one of the first attempts to produce special purpose built hardware solutions to database problems, which were the result of the limitations of hardware and software during 1970s. There were two major advantages claimed for the database machines. They were:

- specialised computers designed for a specific job
- "almost entirely devoid of software" (Banerjee & Hsiao, 1978).

Novel solutions were devised using methods such as specially designed disks with multiple read heads, each with its own microchip, or bubble memory (also known as *zero-rpm* associate disks), to perform parallel searches and achieve higher performance. However, these systems did not gain acceptance because to produce high retrieval rates the hardware was pushed to the limits of its performance, and for most applications software indexing will always beat hardware brute force (Stonebraker, 1994).

The main influence of these early systems was to introduce the notion of backend database processing which opened the way for the wider acceptance of the concept of database servers attached to a client server network (Healey *et al.*, 1998b). This market niche is now growing in importance in relation to the adoption of parallel database servers. Some of the large software vendors are providing parallel database options for their customers, particularly for use in data warehousing and rapid data processing for internet applications. For example, the SAS Institute, which markets data warehousing software as part of the SAS system, has developed a parallel database engine for improved database performance (SAS Institute, 1999; Ghosh, 1999) and is improving SAS performance using parallel processing (Olsen & West, 1999; Williams, 1999), Oracle has implemented a fully parallel database server for corporate use (Oracle Corporation., 1997b) and Microsoft has developed a scalable SQL server that incorporates parallel processing techniques (Microsoft Corporation, 1999).

The potential role of parallel processing in implementing relational database operations has long been under consideration (Bitton *et al.*, 1983) particularly as the relational database language is the only one that lends itself to backend processing. However, as serial processing was the only available technology for general purpose computing, the special purpose database machine was the only platform where the potential for parallel processing could begin to be exploited. Unfortunately, due to the specialist nature of the hardware it meant that the cost could only be justified where the database load comprised a large proportion of the overall processing, and therefore it never gained much popularity as a type of machine for commercial use.

5.1.2 Contemporary Parallel Architecture for Database Servers

5.1.2.1 SIMD

SIMD machines have not proved appropriate for the multi-user, I/O intensive requirements of database processing, because the step by step processing methods are too restrictive. SIMD machines have to execute the same instruction on each processor and this is not useful for database management. Therefore there are no commercially available SIMD database applications, and although a few research databases may exist there is little evidence of this research available.

5.1.2.2 MIMD

There are three types of MIMD architecture that can be considered as suitable for database computers. These are shared memory machines, shared disk machines and shared nothing machines. Shared memory configurations consist of a collection of processors attached to a shared memory bus, each accessing a common shared memory (as described in chapter four). Shared disk architecture consists of a collection of processors with private memory that access a shared disk system e.g. VAX clusters. The final architectures are shared nothing where processors have their own local memory and disks and are connected by a fast inter-connect. They can consist of specialist hardware or comprise a network of workstations connected using ethernet for example. They have become known as software database machines because very little of the hardware is custom made, rather the database software uses the available hardware to its best advantage.

Shared memory and shared disk machines have both been perceived to suffer from significant limitations of bus contention which arises from overlapping traffic on the bus and has the potential to limit the scalability of larger systems (Healey, 1998a, Stonebraker, 1994). This makes them less useful as database servers for very large volumes of data. However, improvements in bus technology including multi-bus and cross-bar switching configurations have gone some way to solving these problems.

Shared nothing architectures, for example work station clusters, do not suffer the constraints of the other two types of architecture because memory and disks are held locally. It does however, require the database software to support a distributed database system to provide communication links to enable the processors in the cluster to share data or database queries. The main advantage offered by this environment is its potential to scale to very large numbers of processors (hundreds, possibly thousands).

5.2 Software

Two main methods of using parallel processing in database software have been identified: the first is to extend existing facilities and interfaces of the database software to allow basic usage of multiple processors on parallel disk systems. The second is to exploit more fully the parallelism available by re-writing the database kernel functions.

5.2.1 Extended Database Software

Database software which makes use of the capabilities of SMP architecture has been available for a number of years e.g. Digital or Sequent. Many of these applications use background processes for database writes or asynchronous read-ahead from disk, and it has been relatively easy to transfer and distribute processes across multiple processors. The techniques provide markedly improved response times for a user process which is confined to a single processor, providing the machine is not fully loaded. They have become very successful because they provide improved performance, while maintaining the ability to run other standard applications on standard operating systems, without the need for recoded parallel versions of the software.

5.2.2 Fully Parallel Database Systems

A number of software vendors have produced MPP (Massively Parallel Processor) versions of their software, these include Oracle (Oracle Corporation 1997b), CA Ingres, Informix and Sybase, to run on machines such as are supplied by Parsys (1991), nCube and Meiko (Beckett, 1995).

The focus of this research project centres on the Meiko Computing Surface. Therefore, this will be used to demonstrate the evolutionary approach to parallel database servers which, according to Holman and Barton(1991), can be explored under three main areas:

- multi-instance support
- parallel lock management
- parallel file servers

A detailed description of the configuration of the Meiko Computing Surface used in this research project can be found in Chapter seven.

5.2.2.1 Multi-Instance Support

Multi-instance support means that each of the processors runs its own instance of the database management software (Figure 5-1). Each instance supports its own set of users and has its own set of background processes. The instance is linked to a single database using the parallel machine inter-connect, and communicates with each of the other instances using the distributed database capabilities of the software. By comparison, an SMP machine will only usually start a single instance of the database management software.

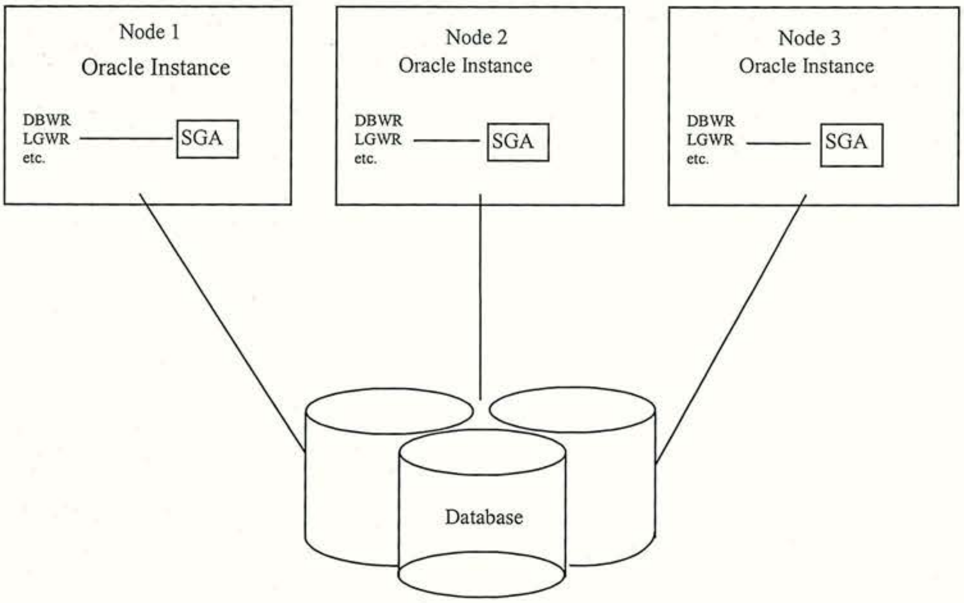


Figure 5-1: Multi-Instance Database System (after Oracle Corporation, 1997b)

5.2.2.2 Parallel Lock Manager

Database locks are a method of synchronising database tasks to prevent simultaneous changes occurring to the same data. Locking is particularly important in an environment where there is the opportunity for multiple users, or processors accessing the same data. The lock manager has responsibility for the locks that are acquired and released in the course of updating the same database from multiple instances. It is implemented using a number of co-operating sequential processes that are located on transputer nodes in the computer. It is important that as the number of nodes and, therefore, instances increases, the lock manager can also be scaled (Jakobek, 1990). If update transactions are held up by the lock manager for longer than necessary, locking can become a performance issue.

5.2.2.3 Parallel File Server

The parallel file server is based on multiple disk controllers each with their own substantial data cache. In a typical configuration this is usually a minimum of 16 Mbytes. The parallel file server allows both data and log files to be striped across a number of disks to improve performance. Striping ensures uniform access over the whole of the parallel system for all instances and so maximises the benefit of concurrent disk I/O and bandwidth (Meiko, 1992).

A number of advantages have been identified for this combination that uses mostly commodity components with specialised items restricted to the disk controller and the interconnects. They include flexibility, scalability, and the provision of an upward path from standard architectures for resource hungry database applications (Healey *et al.*, 1998b). This type of system is also suited to both transaction processing and more complex longer transactions (Newell & Easterfield, 1990), which, is important for use with GIS.

5.3 Parallel Queries

Having considered the hardware and software, it is now necessary to examine parallelisation of database queries themselves. One method is to utilise the distributed database capability to harness the power of multiple instances on different nodes in the processor network, although, it requires significant additional programming by the user. It also generates extra traffic on the internal network, which may very well cancel any advantage gained from processing over multiple nodes.

There are a number of aspects to query optimisation, which centre around how the query is broken down, and the extent to which parallelisation is dependent on query content.

5.3.1 Query Decomposition

Pirahesh *et al.* (1990) described two methods of query optimisation: static and dynamic. Static optimisation implies that the breakdown of tasks between the available processors is decided by the query executor at compilation time. Decisions may be made on a *cost* basis, by considering the different options available and choosing the one likely to give the best performance, similar to traditional query optimisation, but with the additional option that certain parts may be parallelised.

Dynamic optimisation adapts the static plans based on a mixture of run-time information about the loading on the processors and the other system parameters, such as memory availability.

The main drawback of these two approaches is the size of the search space required to find the optimum solution. This could be very severe if dynamic optimisation is employed, resulting in significant resource usage. The burden of query optimisation may become too heavy, particularly under conditions of heavy loading (Oracle Corporation, 1991).

Therefore, a two stage strategy could be adopted. The first stage generates a query plan without reference to parallelisation issues. The resultant plan is then optimised for parallel execution. There is a trade off with this strategy. While it makes for a much simpler solution, it may not find the most cost-efficient search plan so operations which could be decomposed over several processors may have been missed (Healey *et al.*, 1998b).

5.3.2 Query Content

The level of query decomposition is very much dependent on the contents of individual queries. SQL, for example, has a number of clauses and conditions which lend themselves readily to parallel execution; however, some others do not. An obvious candidate for parallelisation is relational joins, where the inherent parallelisation of simultaneous multi-table scanning has been exploited since the advent of database machines. Sub-queries, where the result of one query is used to drive the search conditions for another query, are also susceptible to parallel execution.

Difficulties start to arise for clauses containing grouping operators. These operators usually have to receive and process all of their inputs before the results stream is output, and therefore, cannot proceed in parallel with other operations.

5.4 Implications for GIS

The recent developments in parallel database management systems examined above have a number of implications for corporate GIS applications based on either integrated or hybrid model approaches.

5.4.1 Integrated Model

Corporate GIS applications based on integrated model GISs are in an ideal situation to make use of the vast and speedy data storage and manipulation advantages offered by the current parallel database technology. It is now possible to store the contents of very large digital cartographic databases in fully normalised form using commercial parallel database software, on general purpose parallel machines. As the uptake of parallel processing for *behind the scenes* data management increases, the use of special purpose hardware for databases will disappear, and the requirement for vendor specific GIS file systems for data storage will be redundant. The advantages of parallel processing to businesses running corporate GIS are threefold: a significant reduction in the need for hardware and software requiring specialist support; an improvement in the integration of GIS within the organisation because the GIS applications are able to utilise hardware and software used by other information systems resident in the organisation; and a strengthened business case for corporate GIS because it is able to integrate much more readily with the IT structure in the organisation.

The use of parallel database technology for corporate GIS using the integrated model approach should not require major changes to storage methods already implemented for performance benefits. Standard optimisations such as the storage of map co-ordinate strings in bulk data types will produce much greater performance gains in a parallel rather than a sequential environment, especially if combined with various fetching methods for retrieving groups of rows that are supported by leading vendors. The use of new helical code indexing methods (Oracle Corporation, 1997d) can be expected to provide matching gains in searching performance.

The technical developments have been made on the basis that users will require to handle databases of very large proportions i.e. hundreds of gigabytes. Therefore integrated model GISs built on top of parallel database software are well placed to meet the future demands for the mapping requirements of large organisations identified in the earlier chapters.

5.4.2 Hybrid Model

The hybrid GIS model still dominates the commercial market place, despite the many and growing advantages of the integrated model, and presents a number of difficulties when considered in conjunction with the use of external parallel databases. When implementing a parallel approach to hybrid GIS there are three main areas to consider: the overall suitability of the hardware platform, the effectiveness of the GIS/database interface and, finally, the links between corporate GIS and the organisational information systems.

Of the two main parallel platforms the shared memory multi-processor machines have worked well because of the standard nature of the operating system and the availability of versions of the widely used commercial applications. For distributed memory machines it is a different story. There are no parallel versions of GIS software commercially available. Serial corporate GIS applications can, however, potentially run on individual nodes of distributed memory machines providing they are composed of commodity processors and a standard operating system. Parallel relational database software is available for this type of environment and, providing the interface between the database and GIS application is functional, it is possible to harness the advantages of parallel database technology.

The main issues for hybrid corporate GIS centre around the performance and functionality of the interfaces between GIS software and the external parallel database. GIS like Arc/Info and Smallworld have their own internal data management facilities, providing interfaces to external databases, while others rely totally on external interfaces to manage attribute data. Owing to the nature of these interfaces they tend to be relatively resource intensive. When spanning between systems it is difficult to provide the same level of functionality and performance as that of a relational join within a singe database environment. It may be necessary to separate out the nodes supporting GIS from those supporting the database to avoid overloading them.

5.4.3 GIS and MIS

Linking existing MIS and corporate GIS functions (whichever model approach is used), using general purpose parallel database management software, is likely to present a number of difficulties, many of which were evident in the early 1980s when IT functions were first centralised and based around mainframe computers. The difficulties relate directly to the problem of different loads placed upon the database by different types of query i.e. a long, complex queries vs. transaction processing loads, where the queries are simple but must be performed many times over. GIS queries are often the former, whilst other MIS generate the latter.

For a large utility company, for example, large parts of the MIS would be concerned with customer and billing information, while others would be concerned with asset management and the control of streetworks and repair programmes. A number of tables across the database would have direct and indirect linkages with GIS for digital map based query and display. Meeting both GIS and MIS requirements from a hybrid model system on a parallel machine would result in a very mixed load on the parallel database.

The ability of a parallel database system to handle the conflicting requirements of mixed work loads generated by corporate GIS and MIS will be a key aspect of its success in business because it will enable the corporate data integration that has been long sought after but never quite achieved. There are a number of options available in a parallel environment that are not available to the same degree in single processor environments, e.g. the ability to group processors together for specific functions or user groups, all able to access the same database. These options need investigating.

There is no doubt that the current parallel databases can outperform their standard mainframe counterparts, particularly for transaction processing tasks. However, it remains to be seen how well the system will cope with the mixing of transaction loads and with numerous different types of queries, some GIS based, some not, being streamed in by large numbers of users. Either may seriously impede the performance of transaction processing tasks, or alternatively, will have their performance impeded by these tasks. One of the aims of this research is to explore the effect of mixed workloads from both integrated and hybrid model GIS, and consider some of the options available for improving performance.

5.4.4 Internet

Another key area of particular interest for a GIS/parallel database combination is the internet. The demand for spatial data, particularly in the travel industry, is beginning to grow. Examples of this are tourist information services that offer active spatial content and multi-media inserts, road traffic monitoring services providing up-to-the minute reports on road congestion, accidents and road works, the NHS providing spatial information on local health services including shortest path analysis to health centres, pharmacies, opticians and dental services. Access to the Internet is increasing rapidly as people connect from home, or gain access through Internet cafes or public services such as libraries. Already there are in existence a few spatial data sites with large, parallel servers fielding approximately 40,000 hits a day.

6. Benchmarking

The previous chapter identified a number of areas that require further investigation concerning the use of parallel databases with GIS. These range from more general requirements, concerning the suitability of the hardware platform for running both GIS and parallel database tasks together, the performance characteristics of the parallel database, and the parallel execution of SQL in such an environment, to much more specific details concerning hybrid GIS models using external parallel databases. They include interface issues such as speed, resource usage and flexibility, and work load issues involving the impact of different types of query running in the same environment, accessing a shared database.

In this chapter design considerations are given to the construction of a performance test environment for the use of parallel database management system software with corporate GIS. There are three main areas of focus:

- the indicators to be measured - given the complexity of the system;
- how to measure the chosen indicators;
- the best method of reporting the results.

One method developed to test hardware platforms and software applications systematically, to allow for comparison with other similar systems, is benchmarking. The advantage of benchmark tests is that they provide rules for designing and running tests and also reporting the test results. Although the number of benchmarks for serial systems far outweighs those for parallel systems, examination of individual benchmarks can provide useful information for the construction of performance tests for the parallel GIS/database system in this research. In particular, those indicators which have successfully characterised the system, the structure of tests and methods of reporting are of particular importance, The most important information of all however, is the reported weaknesses and failures of other benchmarks – to avoid the pitfalls that others have already discovered.

The following chapter is split into five sections. The first section is concerned with defining what benchmarking is, what benchmarks are used for and how they are used. This is followed in the second section with a review of existing benchmarks[3] for parallel hardware and operating system analysis. One area where benchmarks have been used and developed extensively is that of database performance, initially to assess on-line transaction processing performance and later expanded to include many different aspects of database performance. In section three four of the more popular database benchmarks are discussed. Section four describes the Wisconsin benchmark. The Wisconsin benchmark was developed initially for serial systems but later adapted for parallel database performance assessment. This section details some of the findings concerning the design of performance tests for parallel databases, including the development of underlying datasets, indicators to measure, and how to measure them. The final section summarises the chapter:

6.1 Introduction

Benchmarking is a method of measuring the performance of one computer system and its software against some pre-set criterion, i.e. other computers in the field of expertise, or a list of requisite performance requirements. The need to measure and compare systems has become more important as the variety and range of computer systems, networks and web technologies have expanded rapidly. All systems have their strengths and weaknesses making them more suited to some particular jobs than to others. Identifying which systems are best suited to a certain defined operation, and how one system performs against another has always been a difficult task. Defining a test that is representative of the task, ensuring that these tests are carried out fairly on individual systems, and that the results truly represent the ability of the system to perform as required - is a path littered with pitfalls (Stonebraker, 1994).

Over the years, benchmarks have become increasingly important because of a consensus of what to measure and how to measure it. The results of benchmark tests are often available in the computing literature and have become very important to the computer industry (Gray, 1993). There is no universal benchmark that can be used to measure the performance of systems on all applications. The performance of a particular system depends upon the nature of the work required. Systems typically designed for a few specific problems may be incapable of performing other tasks, for example, machines configured for transaction

[3] The benchmarks detailed in this chapter are those that were available during the period when the test hardware configuration was available. Appropriate benchmarks developed subsequently will be discussed in chapter thirteen.

processing usually perform poorly when processing complex database queries. Therefore, there exist a whole host of benchmarks targeting particular areas of computing. The type and workload of individual benchmarks are dependent on the function of the exercise. Each benchmark specifies a workload, whether real or synthetic, which characterises typical applications in that particular area. The performance of the workload on different systems gives an estimate of the relative performance of each system on that particular type of problem.

Benchmarks are also characterised by the indicators that they measure, such as CPU, I/O, graphics, and transaction processing, to name a few. They are used by numerous sections of the IT population. Programmers use available benchmarks to choose among design alternatives; product developers compare their products with those of their competitors; and users, not interested in the technical aspects of benchmarking, make use of benchmarks to aid in the purchasing process - indicating the range of performance that might be expected from different hardware and software configurations (Gray, 1993).

There are several organisations that define standard, domain-specific benchmarks, standard price metrics, and standard methods for reporting results. Of these the most prominent are:

- **SPEC** (Standard Performance Evaluation Corporation) - a consortium of vendors defining benchmarks for the workstation environment with particular emphasis on UNIX systems (**http://www.specbench.org**) (SPEC, 1998a, SPEC, 1998b).
- **Ziff-Davis** - who produce benchmark software for PC and desktop environments (**http://www.zdnet.com**).
- **BAPco** (Business Application Performance Consortium) - a consortium of hardware and software vendors defining benchmarks to measure the performance of personal computers in client-server and groupware configurations (Gray, 1993).
- **TPC** (Transaction Processing Performance Council) - a consortium of vendors defining transaction processing benchmarks (**http://www.tpc.org**).

During the 1990s the expansion of the computing market with high performance PCs, networking intranets and the internet has seen a corresponding increase in the number and variety of benchmarks and benchmarketing companies available.

The benchmarks of interest to this research fall into two main areas: those that measure aspects of performance of the hardware and operating system components for parallel machines, and benchmarks concerned with measuring the performance of software systems such as GIS and database applications. The following sections of this chapter concentrate on published benchmarks available for measuring the hardware, operating system and software for parallel machines and database performance.

6.2 Parallel Architecture Benchmarks

The National Physical Laboratory (NPL) (Brocklehurst, 1991) has made an extensive survey of computer benchmarks available to vector and parallel processor systems. The benchmarks have been classified into three separate groups: general parallel processing; distributed memory; and UNIX system benchmarks. Those that fall into the general area of this research project are detailed below.

6.2.1 General Parallel Processing Benchmarks

Four general parallel processing benchmarks have been identified, these are: Dhrystone (Weiker, 1984; Weiker, 1990), Whetstone (Weiker, 1990; Curnow & Wichmann, 1976), Hartstone (Donohoe *et al.*, 1990), Khornerstone (Wilson, 1989).

Whetstone is a synthetic benchmark developed at NPL to reflect workload during the 1970s. It represents a computational mix sampled from a large collection of 60 Algol programs. The benchmark provides an overall measure using different precisions during integer and floating point calculations and returns a measure of kilo-Whetstones per second. Whetstones major defect is that it is difficult to tell whether the code has been tweaked by vendors because there are no precise reporting requirements, and it does not specify the precision of floating point numbers (Dowd, 1993).

Dhrystone is a synthetic benchmark developed in a similar manner to Whetstone but directed to non-numeric systems-type programming. The benchmark was originally written in Ada by Reinhold Weiker and recoded in C by Rick Richardson, and returns a performance measure in Dhrystones per second. It exercises fixed-point computations, similar to those found in compilers or systems programs. There are also a number of defects present in this benchmark, namely, it is an unrealistic simulation which does not capture the full nature of fixed point applications, it fits into the memory of most machines so does not test the effects of paging, and it is sensitive to compiler optimisations (Brocklehurst, 1991; Dowd, 1993).

Hartstone is a real time benchmark written in Ada. It measures the ability of systems to handle multiple real-time tasks efficiently.

Finally, Khornerstone is a mix of public domain (including Dhrystone and Sieve) and proprietary benchmarks. It measures the characteristics of processor, floating point and disk performance for a single user.

6.2.2 Distributed Memory

Two benchmarks have been specifically developed for examining the performance of distributed memory machines. These are: Euroben (Van der Steen, 1991) and Genesis (Hey, 1990; Scott, 1989; Mierendorff & Trottenbeg, 1988).

The Euroben benchmark consists of three modules, written in Fortran, for the assessment of scientific computing on super computers and distributed memory machines. The first module is of most significance and contains architectural benchmarks for memory bank conflicts and communication bottlenecks. The other modules are for testing algorithms and matrix multiplication.

Genesis benchmarks are aimed at evaluating MIMD-DM machines and were constructed as part of the Esprit 2 Genesis project P2702 (Hey, 1990). It is split into two levels, the first consisting of synthetic code fragments, the second containing the core of a molecular modelling routine and a linear equation solver. Most of the benchmarks are message-passing codes written in Fortran77, using mainly double precision mathematics. Version 3 was released in 1994. It contains a few known bugs which are manifest on certain systems, but these are in the process of being fixed (Southampton, 1999)

6.2.3 UNIX Systems

Finally, IOBENCH (1989) is a comprehensive test of data searching and reporting. The program creates files and records with implicit keys. A number of users are created to pick files at random and then write to files at random. Once the file has been picked, the records are read, re-written and finally the users write result records. It was designed at Prime Computer Inc and runs on a number of UNIX systems.

The parallel benchmarks detailed above are used to assess the potential of the architecture to perform complex, processing intensive algorithms, where the whole machine is dedicated to that one task. The y are designed for the purpose of examining the speed of the system as a whole, or the performance of individual components in a single user environment where resource load is, for the most part, predictable.

The requirements of a benchmark for testing the performance of the parallel architecture used in this research project are unusual because the role of the parallel architecture is different. The role is not that of a specialised machine dedicated to any one single type of task, rather to support the varied corporate requirements of multiple users in a speedy and efficient manner.

These parallel benchmarks mentioned do not provide the facilities for detailed analysis of the performance of the MCS where the number of users, processing requirements and data access are all unpredictable. The benchmarks were designed for much older parallel machines, which were usually made of proprietary components and often designed for purely scientific use. The benchmarks often consist of large volumes of code that is platform specific and difficult to translate to another platform. They suffer from two great weaknesses, namely a lack of rigorous design and poor reporting mechanisms, which reduces the use of any benchmark because the results can not be compared with those of other systems.

6.3 Database Benchmarks

The design and execution of benchmarks for databases and transaction processing machines has become such a large subject that it merits treatment on its own. There are several types of query processing that can occur as database transactions, however, this research will concentrate on two: on-line transaction processing and complex query processing. On-line transaction processing is where similar simple queries are processed in rapid succession. The emphasis with this type of transaction is on bulk processing, that is, the largest number of transactions that can be processed in a given time. They are usually measured in transactions per second (tps). The second types of query are complex queries. They are much more *ad hoc* in nature and often require data from many tables to be linked together, usually containing multiple conditions that must be resolved to provide a result. Individual queries can take hours to return results, therefore, emphasis is placed on how to optimise various aspects of the query, for example, the efficiency of indexes and time taken to parse the query, to make the best use of resources to reduce the time taken to execute the query.

The next two sub-sections describe some of the benchmarks that have emerged for both on-line transaction processing and complex query processing. The majority of these benchmarks are designed for serial machines and may not easily translate to the parallel environment.

6.3.1 On Line Transaction Processing

The three most famous benchmarks for transaction processing are those defined by the Transaction Processing Performance Council (TPC), TPC-A (TPC, 1989), TPC-B (TPC, 1990) and TPC-C (TPC, 1992). The need for these benchmarks arose from developments in the 1980s. There was an expectation that high performance transaction processing machines would be in great demand as the proliferation of automatic teller machines (ATM) and automatic petrol pumps (controlled by credit and debit cards) would become very popular and overwhelm computers with huge numbers of transactions (Serlin, 1993). Although the petrol pump deluge never materialised in this country, the sharp increase in transaction processing requirements from ATMs focused attention on the performance of transaction processing systems and in particular how to measure the performance (provoking debate about 1K tps systems[4]) (Good *et al.*, 1985). The competition between high performance

[4] those capable of sustaining 1000 transactions per second.

transaction processing machines aiming for 1K tps, and the medium range on-line transaction processing (OLTP) machines, which relied on high performance micro-processor technology, was fierce. The competition led to manufacturers publishing tps ratings for their particular systems to encourage buyers. The drawback with this was that there was no agreement on exactly how tps ratings were to be derived. Each vendor devised their own suite of tests to suit the market they were aiming at and, of course, the idiosyncrasies of their own systems. Very few details of the tests were ever published as they were regarded as confidential so there was never any sense of whether the tests were actually comparable (Serlin, 1993). Leaked results of the tests often caused benchmark wars where vendors would attempt to outdo each other by producing better numbers.

The confusion and doubt that existed over the reliability of vendor devised tests spurred the development of publicly available benchmarks that could provide a true comparison between various systems. The Debit/Credit benchmark, the first of these benchmarks, was closely followed by TPC-A TPC-B and TPC-C - from the TPC. Other benchmarks, such as the Wisconsin benchmark also gained popularity and this benchmark was also used to measure the performance of parallel databases. The following subsections detail the Debit/Credit benchmark and those from the TPC.

6.3.1.1 Debit/Credit Benchmark

The Debit/Credit benchmark (Anon *et al.*, 1985, DebitCredit, 1986) was one of the first to specify a true system level benchmark where network and user interaction components of the workload were included. The benchmark specified that the total system cost should be published with the performance rating to allow for true comparison of results between different platforms, a practice that has become a fundamental part of any benchmarking exercise. The rating was described in terms of high-level functional requirements rather than specifying hardware/software platforms or core-level requirements facilitating comparisons between results obtained from different platforms.

The benchmark also included workload scale-up rules where the number of users and the size of the database tables increased proportionally with the increasing power of the system to produce higher transaction rates. Finally, the benchmark specified that the overall transaction rate would be constrained by a response time requirement, that 95 percent of all transactions had to be completed in less than one second (Shanley, 1998). A number of features specified in the Debit/Credit benchmarking process were later incorporated in the TPC benchmarks.

6.3.1.2 TPC-A

The TPC-A benchmark exercises the system components necessary to perform tasks associated with on-line transaction processing environments performing update-intensive database services. These are characterised by multiple on-line terminal sessions, significant disk I/O, moderate system and application execution time and transaction integrity.

The benchmark is based on the transactions of a hypothetical bank. The bank has one or more branches and each branch has multiple tellers. The bank has multiple customers, each with an account. The database represents the cast position of each entity (branch, teller and account) and a history of recent transactions run by the bank. The transaction represents the work done when a customer makes a deposit or withdrawal against their account. The transaction is performed by the teller at some branch (TPC-A, 1989).

The benchmark can be run in both local and wide area network configurations. It uses a single simple update-intensive transaction to load the system under test, intended to reflect an on-line transaction processing application, but does not reflect the entire range of requirements typically represented by multiple transaction types of varying complexities.

6.3.1.3 TPC-B

TPC-B represents the batch version of the Debit/Credit benchmark, without the network and user interaction factored into the workload. This benchmark has been favoured by hardware companies who sold servers and database companies because they felt it represented the customer environments they sold into. TPC-B uses the same TPC-A transaction type (banking transaction) but without the network and user interaction components to leave a batch processing benchmark (Shanley, 1998).

6.3.1.4 TPC-C

TPC-C is a much more complex benchmark than TPC-A. It includes multiple transaction types, more complex databases and overall execution structure. The benchmark portrays the activity of a wholesale supplier, simulating a complex environment where a population of terminal operators executes transactions against a database. The benchmark is centred around an order-entry environment where transactions include entry and delivery orders, recording payments, checking the status of orders and monitoring the level of stock at the warehouses.

The benchmark specifies the number of different types of transaction that occur during execution. These range from on-line transaction processing type transactions that consist in recording a payment received from a customer, to less frequent transactions where operators request the status of a previously placed order, process a batch of 10 orders for delivery, or query the system for potential supply shortages. A total of five different types of transaction are used to model the business activity.

While the benchmark is based on the activities of a whole sale supplier, it could also be used to simulate the activity of a wide variety of business sectors that must manage, sell or distribute a product service, making it a much more flexible benchmark for the business sector.

6.3.1.5 Advantages and Disadvantages of TPC Benchmarks

The benchmark tests described above provide a number of insights into the construction of performance tests for the parallel database environment. The first are the standards they have set regarding the structuring of tests and the strict reporting of results, with an insistence on public reporting. As the performance tests will be performed on more than one platform consideration must be given to how the performance tests are run, what is measured, how it is measured to allow for comparability of results. Test results themselves should be presented in a consistent manner and the method of reporting clearly detailed. Secondly, the tests provide insights into constructing tests for distributed systems where multiple users access services across networks, allowing the influence of network speed and network traffic can be accounted for. Finally, the tests provide details on constructing scalable tests where either work volume or user volume is increased.

More specifically, they describe in detail techniques for testing different aspects of databases from the design of database tables and indexes to measuring the performance of database queries. The individual benchmarks have their own strengths. The principal strength of TPC-A is that it is an end-to-end system benchmark that exercises all aspects of an on-line transaction processing system. It is of importance to the research because it represents a system with users at terminals conducting simple transactions over a wide area or local area network to a database server, and based on a computing model that is easily understood (Shanley1998).

TPC-B is designed to stress test the core portion of the database system that performs transaction processing. It stresses CPU by generating streams of transactions as fast as possible, and the memory and disk I/O subsystem by causing large numbers of small read and write jobs. They require that a system demonstrate that it can meet the reliability and security features of the ACID (atomicity, consistency, isolation, and durability) tests by detailing three separate durability tests (Hanson, 1998).

TPC-C is a very useful benchmark because it measures the performance of the systems when multiple transactions of different natures compete for system resources. There are a mix of five concurrent transactions of different types and complexity. The benchmark provides details for an increased complexity of the data structure used which allows a greater diversity in the data manipulated by the different types of transactions. The data entered by operators in TPC-C include some of the basic characteristics of real-life data input, and operators may enter invalid data which would force the transactions to be cancelled.

The TPC-C benchmark is a scalable benchmark, which requires a system to demonstrate its ability to increase the number of terminals used and the size of the database proportionally to the computing power of the measured system (Raab *et al*, 1998).

Unfortunately the TPC benchmarks do not provide a suitable suite of benchmarks to performance test the parallel database system used in this research. TPC-A and B are based exclusively on transaction processing tasks and the ability of the system to sustain very high data throughput for long periods. Although the performance tests include the requirement to test the system when transaction processing is running, the aim of the tests is to examine its effect on the performance of other tasks occurring simultaneously, rather than gauge the tps rate. TPC-C provides a good foundation for constructing performance tests. However, this particular benchmark is modelled around an order-entry environment where movement of goods from one location to another is the cornerstone of the model. The model for the performance tests in this research is far more sedentary where large numbers of users are accessing data from the database using GIS as the access tool. Finally, while the benchmarks provide many useful features, all three are designed for serial machines and would have to be adapted for use with a parallel database.

6.4 A Parallel Database Benchmark

The Wisconsin benchmark (Bitton *et al.*, 1983) is one of the few database benchmarks that has been adapted for use with parallel database systems. The Wisconsin benchmark was designed to test the performance of the major components of the relational database system, by providing clear details on the semantics and statistics of underlying tables. A good understanding of the structure of tables made it easy to add new queries to the model and understand their behaviour.

The Wisconsin benchmark is very useful because it was designed to be flexible and allow a wide variety of update and retrieval queries, and was therefore not restricted to transaction processing only. It also used synthetically generated tables that were defined in the model. Synthetic tables were used for two reasons. Firstly, real data is very difficult to scale and secondly it is difficult to accurately specify selection queries that return a percentage of data. Difficulties in controlling the environment are increased where relational joins are involved because there are too many unknown factors to be able to guarantee that queries will return results of a specified size. The Wisconsin benchmark was used to evaluate a number of database systems running on parallel machines, including Teradata (DeWitt *et al.*, 1987), Gamma (DeWitt *et al.*, 1988; DeWitt, 1990), Tandem (Englert *et al.*, 1989) and Volcano (Graefe, 1990). The benchmark was used to measure the speed-up and scale up characteristics of the above systems (DeWitt, 1993). As described earlier in chapter five, speed-up indicates whether adding additional processors and disks to a database system will result in a corresponding decrease in response time for individual queries. Scale up measures whether a constant response time can be maintained as the workload increases, by adding a proportional number of disks and processors.

Scale up and speed-up experiments were carried out on the parallel database systems mentioned above using a subset of the selection and join queries from the Wisconsin benchmark. Using tables of 100,00, 1 million and 10 million tuples respectively, two copies of each table were created and loaded and the results of the queries stored in the database (to avoid the speed of the communications link becoming a factor).

The Wisconsin benchmark proved a useful model for measuring metrics such as scale up and speed-up when considering the effect of increasing processors or disks in parallel database systems. The main measurement used was that of response time, although it was noted that it is important to establish whether response times measured could be constrained by the speed of the communications network.

When designing benchmark tests to measure speed-up and scale up for parallel database systems there are several issues to consider. Firstly, benchmark tables must be chosen carefully. They must be large enough to ensure that the number of levels in each index remains constant as the number of disks over which the tableis declustered is increased. This is to avoid artificially producing super-linear speed-up. Secondly, when setting up a database for measuring speed-up, the inter-connect communication between processors must be considered, particularly if the base measurements are taken using a single processor system. When moving from using a single node, where no inter-processor communication occurs, to two nodes the level of traffic may increase substantially depending on the query type and method of execution and the internal network paths.

The performance tools available for serial processors are not available yet in the parallel environment, and therefore it is virtually impossible to monitor what loads are occurring on multiple nodes at the same time. With machines such as the Meiko Computing Surface work is occurring at three different levels in the hardware - the SPARC processors, the transputing processors and parallel disk farm.

The experiments and results reported from the Wisconsin benchmark used on parallel databases are very useful because they provide some insight into performance testing a parallel database, even if the research is not directly applicable to the research project.

6.5 Summary

An examination of existing benchmarks for measuring the performance of parallel machines and database systems has highlighted a number of general features in need of consideration when designing performance tests, and a group of specific features necessary to consider when measuring the performance of relational database systems.

The first, and perhaps most important, consideration is that of the reporting requirements of the benchmarks or performance tests. One of the main reasons for creating performance tests or designing benchmarks is to describe in some measurable and comparable way the performance characteristics of the system under test. The end use of the results will dictate how the results are collated and reported, therefore it is necessary to decide from the outset the main aims of the performance tests, the audience the tests are intended for and the most useful format for presenting the results.

The second general requirement for benchmarks or performance tests is that they are a realistic and fair representation of the system and work under test. By under or over estimating the complexity and volume of work undertaken by the system, a misleading picture of its performance is created which has great potential to hide both the systems strengths as well as its weaknesses. For the purposes of research, where weaknesses are often sought out specifically to allow a thorough exploration of solutions to improve the system, it is necessary to design tests that truly represent the work of the system. For example, many GIS datasets are too large to fit into memory. It is possible to subdivide the dataset such that each chunk is small enough to fit into memory (memory access being faster than disk access). However, due to the inter-connected nature of each of the GIS data structures (points, lines and polygons) in the dataset it is unlikely that individual data structures will be contained within a single chunk. Therefore, the system will often have to work with two or more chunks at a time, making it unsafe to assume that because the data has been divided into memory sized chunks all work will be done using memory alone. Designing tests for this system using data that always fits within memory would give a very misleading view of the system.

The third general requirement is a decision on the intended client base as this will dictate how the performance tests are written. If they are designed as a universal set of tests for use with many different systems the code must be portable and work with the same efficiency on all systems it is loaded on to. However, if the tests are written for a one off system, problems of compatibility over many platforms are irrelevant and the code should concentrate on specific features pertaining to the system under test.

An examination of individual benchmarks for benchmarking database systems highlighted specific aspects of performance tests that require consideration. The choice of indicators to measure will depend on the nature of the tests. Indicators such as, CPU usage will give an indication of overall work load for the system, monitoring the use of paging will indicate whether there is a memory usage problem, and measuring disk I/O will indicate whether reading and writing data to disk is causing a bottle neck.

The metrics often used to measure parallel database systems are speed-up, scaleup and elapsed time. Both speed-up and scaleup are used to measure the effect of adding more resources to the parallel machine, for example, processors or disks. Elapsed time is used for indicating the response of the system to the defined workload. Elapsed time can be used to gauge the time a user will wait between submitting a request to the GIS and the data arriving back on screen to give a measure of acceptability of overall machine performance. Elapsed time can also be used to measure individual elements of the overall machine performance to identify where most of the time is spent, to indicate areas for further research.

A fundamental component of a database benchmark or performance test is the design and structure of the underlying database. One consideration is the use of 'real' data (i.e. extracts of data that would typically be used with the system) or synthetic data (designed and created specifically for the tests). Real data has the advantage that it is truly representative of the work undertaken by the system, and full of all types of anomalies associated with real data. However, it is difficult to design tests using such data because it is difficult to predict the workload generated by any given query, thus making performance measurement much more difficult, particularly when measuring scaleup effects. Synthetic data allows the precise definition, construction and execution of performance tests, but is unlikely to be a fair representation of the data normally used with the system. The use of synthetic data can highlight areas of weakness in the system, particularly, where data manipulation and interface issues are of concern. Real data can highlight tuning issues with the database where specific data combinations are causing an inefficient use of resources. A second area for consideration is the design of the data model. The design of the data model is in turn dependent on the types of transaction required. There are several types of transaction ranging from the short, on-line processing type to highly complex queries where many tables of data are accessed in order to return a result. A data model to support on-line transaction processing will require a few limited tables with one or two simple columns in common with each other. A data model to support complex query processing will itself be highly complex allowing complicated and lengthy joins to occur requiring many tables of inter-linked data. Therefore, it is important to establish the different combinations of performance tests required and consider basic database query structure before designing the data model to be sure the database will facilitate all of the performance tests envisaged.

The design of a series of performance tests for a parallel GIS/database can be broken down into a number of areas. These are, the hardware and software applications used, the type of data used (real/synthetic), the design of the database queries, design of the data model, and finally the reporting mechanisms used to report results. The design, implementation and results of the performance tests will be reported in the following chapters.

SECTION 3

Developing a Test Environment

The problems identified in chapter five concerning the use of commercially available parallel database management software and the popular hybrid model require further investigation. This is especially so, if a parallel GIS environment is to prove a viable solution for the technological and processing problems identified in chapter two. These fall into three main areas of investigation: the performance of the database when used for GIS purposes only, the performance of the parallel system when both GIS and MIS applications are competing for the same tables, and the performance of the GIS/database interface.

The use of a parallel database in either an integrated or hybrid model requires the speedy retrieval of information from the database, particularly if the interfacing linkages between the GIS and the database are not sufficiently efficient or as flexible as one would like. To reduce the effect of the interface bottleneck it would make sense to have the database perform the majority of the work, and only pass back the subset of data required by the GIS for further analysis or display. It is therefore important to measure the database responses to the requests it receives. These include increasing the work load, the effect of different indexing strategies, and the effect of storing data blocks in memory (known as the buffer cache) on retrieval times.

The second area of interest concerns the performance of the system when both GIS and MIS are in operation. There are several strategies that could be conceived to overcome the problems. However, it is important first to gauge the effect that conflicting query types have on database performance before considering what solutions to employ. To that end it is necessary to devise a series of tests to replicate this particular environment to measure the effects of performing both long transaction, complex queries and short, transaction processing type queries simultaneously on the same database.

The final area of interest is the effect of the GIS/database interface on the performance of a parallel GIS environment. There are a number of methods for retrieving data from an external database provided by the GIS applications and various strategies involving the mechanics of how the data are retrieved and used that can be explored in relation to the interface.

7. Developing the Test Environment

7.1 Introduction

The development of a test environment is crucial for performance testing because provides a means of exploring the issues of configuration and functionality described in the previous chapter. The test environment encompasses not just the hardware, software, and data, but also all of the decisions made about the testing process. For example, what are the testing aims, who are they aimed at, what level of detail is required, what is being measured and how. Based on the results of those decisions the approach detailed in Figure 7-1 was used.

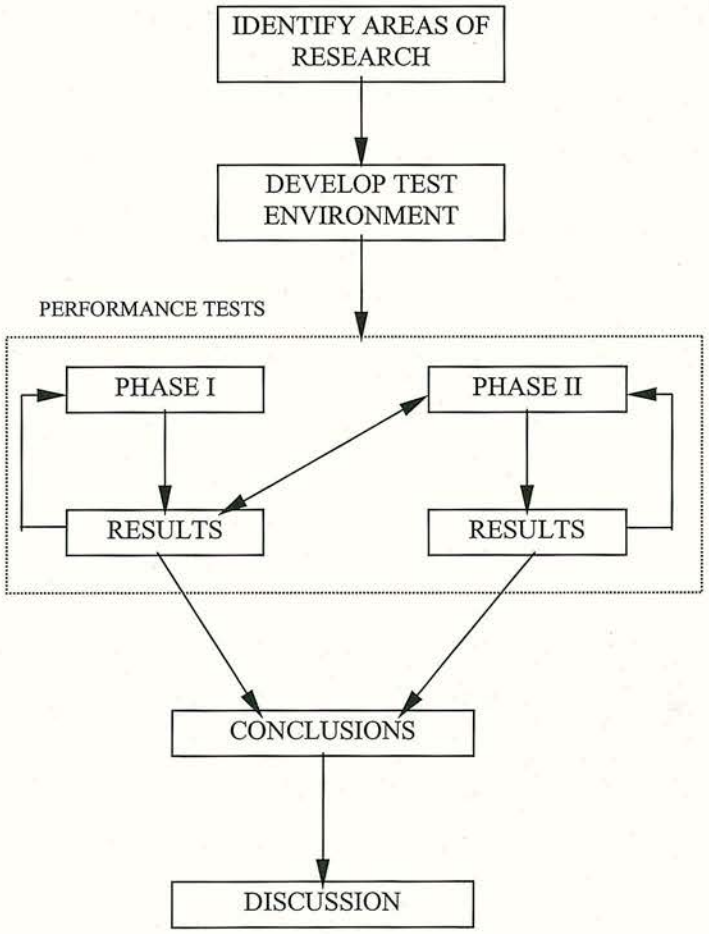


Figure 7-1: Flow Diagram of Performance Test Design and Analysis

7.1.1 Project Design

The project was broken into three phases, a development phase, and two parallel GIS system testing phases called Phase I and Phase II. The development phase involved all the initial loading and configuration of hardware and software, time spent learning to use the software, and trials for devising performance testing harnesses - these were developed to improve the presentation and collection of performance statistics for individual performance tests.

Phase I was used to test the database software more fully by building a pilot database to explore workings of the software and to develop a series of tests for data loading performance, indexing and buffer cache effects.

Phase II introduced a much more realistic database of approx. 15 million rows of real data, simulating a corporate database, based on a gas utility company. This database was used to test the environment for its response to mixed transaction queries and also the database interface.

7.1.2 Performance Tests

The aim of the performance tests carried out in this research was to examine the performance behaviour of the parallel GIS environment to locate the strengths and weaknesses of the system under a number of different conditions. The limitations of the system identified by the series of performance tests gave an indication of the types of benchmarking tests that should be developed for this type of platform for use as a corporate GIS environment.

The performance tests included:

- gathering performance statistics for the individual nodes on the MCS
- comparing the performance of nodes on MCS with those of other mainframes available
- investigating the effect of the cache on the parallel database
- connecting a single GIS to the parallel database
- connecting multiple GIS to the parallel database

Performance tests were developed for both phase I and phase II. The tests were designed for the analysis of specific areas of the parallel database system, and required the development of several new methods for observing, collecting and analysing the results.

Phase I describes the development of the test environment to allow the development of benchmark tests to characterise the response of the parallel database system under different conditions, data loadings and configurations, to attempt to determine optimum database settings and queries. The results from this phase have a twofold purpose: they were used to compare the performance of the parallel database system against more conventional, mainframe, RDMBS, and secondly, provided a baseline for the comparative tests in phase II.

Phase II, saw the development of a more realistic test environment based on two types of corporate GIS, using both the integrated and hybrid models, for extensive testing with more realistically large data volumes, realistic GIS queries and variable transaction processing loads.

The following chapter describes the component parts used in the development of the test environment (see Table 7-1). The following sections provide detailed information about the hardware and software configurations, operation and limitations in an experimental parallel environment.

Table 7-1: Parallel Environment Components

Component	Vendor	Description
Meiko Computing Surface	Meiko	MIMD Computer with 10 SPARC nodes, 16 transputer nodes
Oracle RDBMS v6.2	Oracle Corporation	Multi-Instance version of Oracle designed for MCS
Arc/Info v6	ESRI	GIS commonly used in corporate environment
Smallworld v1.8/1.9	Smallworld	GIS designed originally for Utilities, used to create spatial database for performance testing

7.2 Meiko Computing Surface

The Meiko Computing Surface (MCS) used in this research is an example of a massively parallel distributed system (MIMD-DM). The MCS consists of a small number of relatively powerful SPARC processor chips and a larger number of the less powerful transputer chips which are connected together using an internal high performance communications medium called the Computing Surface Network (CSN) (see
Figure 7-2).

There are several reasons why this particular parallel computer was chosen. Firstly, it uses readily available industry standard hardware and software such as SPARC processors and a standard UNIX operating system i.e. SunOS. It is important that the parallel computer consists of elements that are already familiar and heavily used within the business community. Uptake of parallel computers is far more likely if an organisation is able to use and support the equipment without having to completely re-write applications. Existing software developed to run on UNIX systems under SunOS will run on individual SPARC nodes on the parallel platform with no need for modification, while new software developed specifically for the MCS can take advantage of the parallelism available.

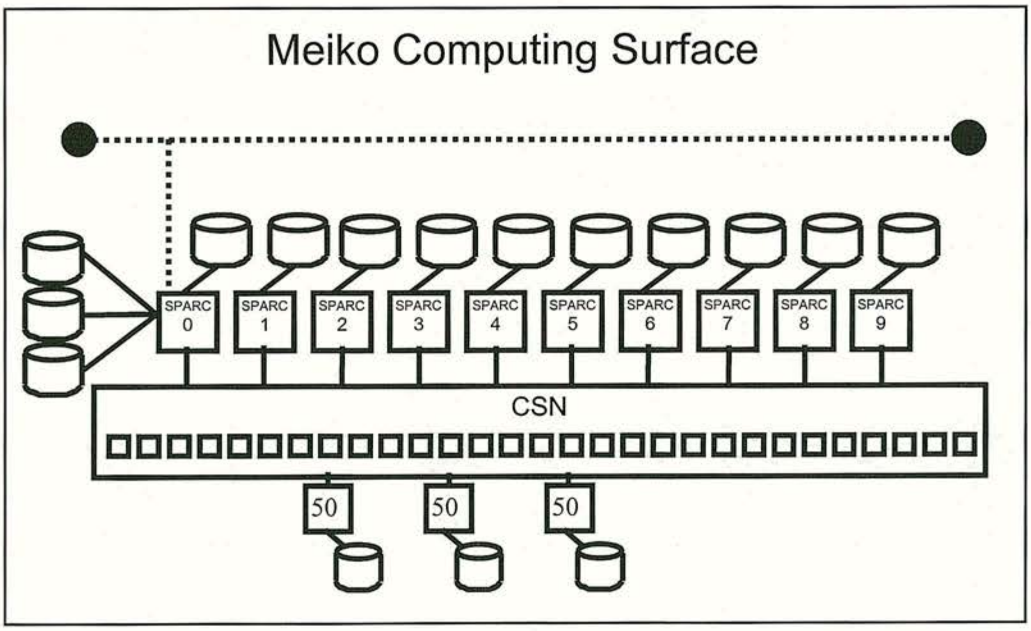


Figure 7-2: Meiko Computing Surface

Secondly, it is thought to be fully scalable i.e. as *N* more nodes are added the parallel application should run *N* times faster. Scalability depends very much on the implementation of individual applications, and is as yet untested using a combination of GIS and parallel database. Scalability is an important aspect for organisations because it means that they can begin by using a modest sized parallel computer and increase the processing power as the size and complexity of the spatial applications and data increase. This gives a good return for money invested and allows the organisation to expand its computing ability as necessary.

Thirdly, the computer is very flexible. The SPARC processors (known as nodes) can be configured in a number of ways to suit the requirements for individual businesses to complement their structure and function. Each node can be used individually, simulating a networked workstation environment where individual users run a variety of applications. The MCS can be partitioned with nodes grouped together and dedicated to particular groups of users or applications. This strategy may prove to be advantageous when several large CPU or I/O intensive jobs are running simultaneously. The whole computer can also be used as a single parallel computer to increase the performance of CPU and I/O intensive jobs. Finally, a version of Oracle (v6.2), a well known, commercially available, database management system, has been designed to run on this parallel architecture.

7.2.1 Meiko Computing System Components

- **SPARC nodes (MK083)** - These are relatively fast Twinhead chips running at 40MHz. The SPARC nodes with their disks effectively form a number of fast SUN workstations clustered together with a much faster network inter-connect than is usual, known as the CSN. These nodes are responsible for the majority of the processing.
- **T800 transputer nodes** - these are less powerful chips, mounted on MK060 quad transputer boards, and are responsible for running Oracle support services, i.e. all the background processes for the Oracle database.
- **Computing Surface Network** (CSN) - the CSN provides inter-processor communications irrespective of the location of the processor. It abstracts all details of the communications hardware and provides a uniform interface to message passing for all processor types. It is a combination of software,

communications processors and custom message routing chips which support the transmission of multiple concurrent messages.

- **Fat Links** - these are multiple transputer links from the SPARC nodes used by the CSN.
- **Disk Controller** - the MK050 disk controller is responsible for the parallel disk farm used by the Oracle database. The disk controller has its own buffer cache of 16 Mbytes. Therefore, the disk controller can store data blocks, increasing the overall amount of data stored in memory in the database system. The buffer cache should improve database retrieval performance by reducing the number of disk accesses for data.
- **Parallel Disk Farm** - a group of disks used specifically by the Oracle database for storing data, metadata and logging information pertaining to the database. The disks work as a unit and are controlled by the MK050 disk controller. They support a flat filing system (FFS) to allow the rapid retrieval of data.
- **Flat Filing System** (FFS) - FFS is a flat filing system structure with no directory structure to allow efficient striping of data across several disks used by Oracle. The striping is done by the Oracle parallel file server, a part of Oracle, and is not available to other applications. The FFS is not actually visible to UNIX except throughout the use of a utility - `fsutil`. The `fsutil` utility allows control of FFS files, including their import to and from UNIX. Backups of Oracle data can be done through `fsutil` or alternatively, made using the `dbcopy` utility. It is necessary to be a member of the `dba` group in order to use `fsutil`.

7.2.2 MCS Configuration

During the life cycle of the research the configuration of the MCS changed as the nature and progression of the research demanded. There were two major incarnations of the MCS and these will be described below. The first was called *Edin* and was used during the initial stages of the project while disk, processor and communication configurations were being decided. The second was called *Hydra*. Hydra was a greatly expanded system and was used for the main performance testing exercises.

7.2.2.1 EDIN

Edin consisted of three SPARC nodes `edin0, edin1 and edin2` respectively, running a variant of SunOS, the SUN version of the UNIX operating system, and eight T800 transputer nodes. There were three disks supporting UNIX file systems and three disks supporting a flat filing system (FFS) dedicated to Oracle (the RDBMS) use and controlled by a MK050 board (Figure 7-3).

The Meiko interconnect CSN linked the three SPARC nodes and transputer nodes, using links from the transputers and *fat links* from the SPARC boards. There was a single internet route to the outside world and this connected from `edin0`. The nodes `edin1` and `edin2` were diskless clients, with all three disks allocated to `edin0`.

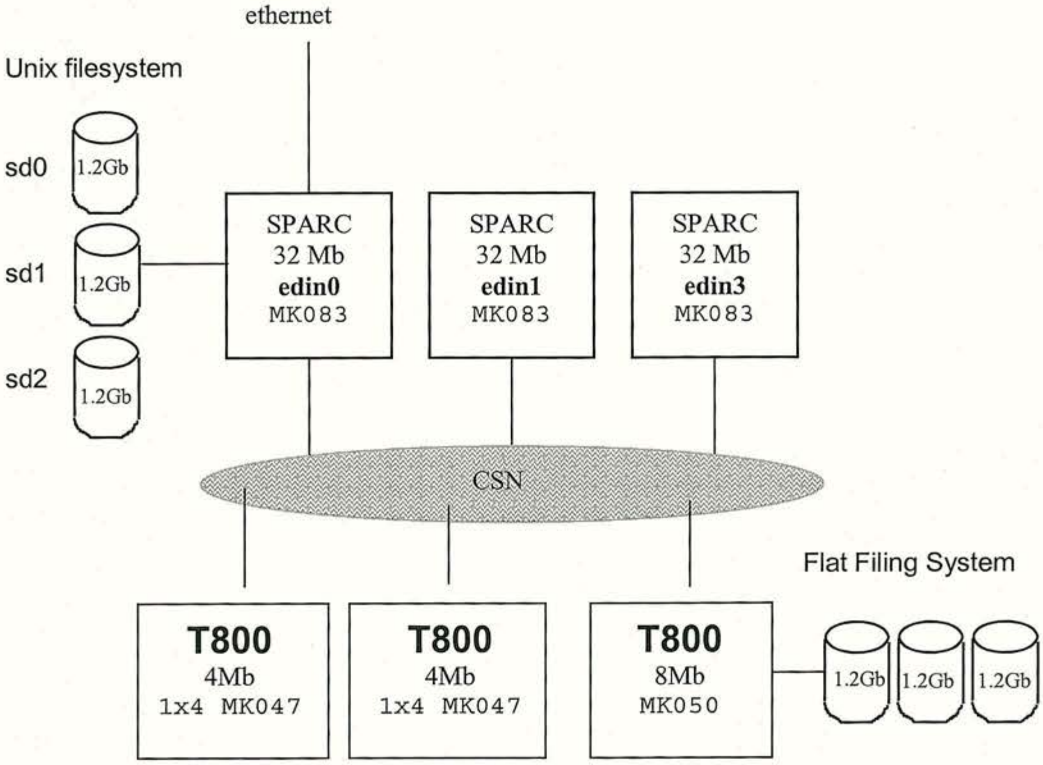


Figure 7-3: Architecture of Edin

7.2.2.2 UNIX File System Configuration

The three UNIX disks `sd0, sd1 and sd2` were mounted as follows (Table 7-2):

Table 7-2: File Locations on Edin

File system	Kbytes	Mounted on
/dev/sd0a	15759	/
/dev/sd0g	183520	/usr
/dev/sd0d	29960	/export
/dev/sd0e	66775	/export/swap
/dev/sd0h	593170	/localhome
/dev/sd1h	903427	/localhome/disk1
/dev/sd2h	903427	localhome/disk2

The GIS packages and the Oracle database had the home directories tabulated in Table 7-3:

Table 7-3: Application Home Directories on Edin

Package	Package account userid	Home directory
Oracle	oracle	edin:/localhome/disk/oracle zircon:/disk/src/master/oracle
Arc/Info	arcinfo	edin:/localhome/disk2/arcinfo
Smallworld GIS	smalwrld	edin:/localhome/disk2/smalwrld

The `oracle` account had a home directory on Edin but also had a home directory on the ZIRCON server where a standard SunOS version of Oracle is installed. This allowed the database to be tested in single user mode, running on a single processor environment for setup purposes.

Smallworld GIS required a file server `swmfs` and Arc/Info required a licence manager `lmgrd` (which in turn spawned `ESRI`) which were created at start-up by `/etc/rc.gis` which in turn was called from `/etc/rc`.

7.2.2.3 The FFS

The three disks controlled by the MK050 disk controller were for Oracle use and supported FFS. The striping was done by the Oracle parallel file server through the utility `fsutil`. Using `fsutil` the following information about the disks was obtained (Table 7-4):

Table 7-4: fsutil report using command 'fsutil devs'

Device	Type	Blocks	Size	MirrorID	State	File System
0	IMPRIMIS 9601-15	2031704	512		+	dbs0
1	IMPRIMIS 9601-15	2031704	512		+	dbs1
2	IMPRIMIS 9601-15	2031704	512		+	log0

This states that at that particular juncture there was currently one file system supported per device, with names `dbs0`, `dbs1`, `log0`. All FFS files are named according to the system *filesystem:filename*. Striping required file systems to have the same alphabetic stem, therefore striping could occur across `dbs0` and `dbs1` but not `log0`.

7.2.3 Hydra

Early in the project a number of additions and alterations were made to Edin and Hydra was created. Hydra initially consisted of seven 32 Mbyte SS-2 SPARC nodes[5] but was quickly upgraded to 10 nodes - `hydra0, hydra1 .... hydra9`, running SunOs v4.1.3. There were a number of disks attached to the SPARC nodes (Table 7-5):

Table 7-5 :Disk configuration on Hydra

Node	No of Disks	Size
hydra0	2	1Gbyte (server)
hydra1	1	2Gbyte (incl. local swap)
hydra2-hydra9	1	1 Gbyte (incl. local swap)

The number of transputer nodes was increased to sixteen, and later to twenty-four. These are as follows:

2 * MK050 @ 1*8MByte T800
4*MK047 @ 4*4Mbyte T800
1*MK060 @ 4*4Mbyte T800

[5](MEIKO MK083 boards)

To summarise, the development of the parallel database environment occurred over a period of several months and was subject to a number of technical changes before arriving at the final configuration known as Hydra. There were three main areas of change. The numbers of SPARC processors were increased from three in the first instance to ten in the final arrangement. The numbers of transputers were upgraded from eight to twenty-four to provide more processing power for background processes in Oracle. Finally, the numbers of disks available were increased from three to eleven. The locations of disks were adjusted as more SPARC nodes were added and application software was installed. Initially all disks were located on node 0 while initial testing was undertaken, as the system expanded the disks were distributed to individual nodes to provide local storage. This greatly reduced the level of traffic on the internal network and reduced the likelihood of bottlenecks occurring.

7.2.4 Oracle v6.2 for the Meiko Computing Surface

This section describes the architecture of multi-instance Oracle RDBMS v6.2 on the massively parallel MCS. The database is designed to exploit the whole of the MCS to allow multiple database tasks to run in parallel and minimise or eliminate bottlenecks in the system. There are three separate parts to the database (see Figure 7-4):

- instances located on SPARC nodes
- background processes located on the transputer nodes
- the disk farm supporting FFS.

The database supports multiple users on each of the processors, by allowing multiple gateways (or instances) into a single database server. Each instance executes independently of other instances. Instances can be located on separate processors or multiple instances can exist on the same processor. They provide the support to co-ordinate all parts of the parallel system, creating a framework for multi-users in a multi-processor environment. There are several obvious advantages of this arrangement. The first is that the multi node system can support many more users than a single node system, all sharing the same resources. Secondly, it offers some protection against failure: if one of the nodes fails, processing can continue on another, thus making provision for non-stop operations, known as 7/24[6]

[6] These are operations that run seven days a week, twenty-four hours a day.

operations that are required to run constantly (Stonebraker, 1994). Thirdly, the architecture allows the separation of functionally distinct applications, for example, development and production systems without restricting access to data.

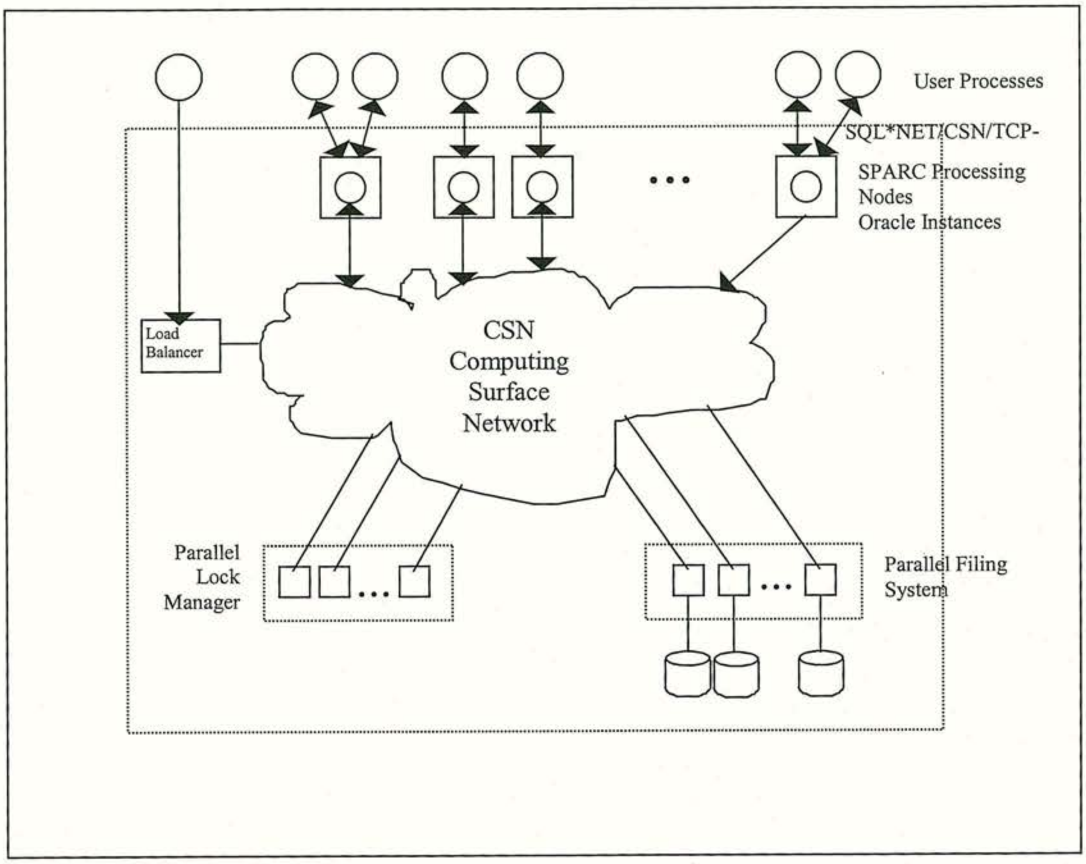


Figure 7-4: Multi-instance Oracle on the MCS (after Holman & Barton, 1991)

7.2.4.1 An Oracle Instance

An Oracle instance is a multi-user software mechanism for accessing and controlling the database (Oracle Corporation, 1991). Each instance includes the following:

- a single shared memory area, called the System Global Area (SGA)
- background processes that are shared by all users

Each instance of Oracle has an *init.ora* file that contains the start-up parameters for allocating the memory area and starting the background processes. On Hydra all instances had the same start-up parameters. Therefore, a single global *init.ora* file was created and referenced by individual instances at start-up.

Multiple instances can run concurrently and independently. They can access the same database simultaneously in shared mode. However, an instance can only access one database at a time. Up to 72 instances can run concurrently on the MCS at any one time.

Each instance consists of a number of components, a System Global Area (SGA) and a number of background processes.

System Global Area

The SGA contains data and control information for individual instances. It is created when an instance is started up and removed when the instance is shut down. It consists of a set of shared memory buffers that contain data and control information. There are two buffer areas, a database buffer pool and a redo log buffer. Two types of blocks are held in the database buffer pool, these being data segment blocks and rollback segment blocks.

During a database transaction data blocks are brought into the SGA and held there. If the data blocks are already in the buffer they are located and held, if they are not there they are fetched from disk and held. During the transaction modifications made to data blocks are noted and stored in a rollback segment block to allow the database to be rolled back at a later date if the occasion arises. Information about the data block and rollback block is stored in the redo log buffer as a redo log entry and eventually written to disk.

Data blocks are stored in the buffer on the basis of a Least Recently Used list. When the buffer becomes full the oldest buffers are written to disk and replaced by the newest ones (Oracle Corporation, 1990).

The operation of the buffer cache is of very great importance to the performance of the database because the time taken to read data blocks from memory is significantly less than that required to retrieve them from disk. By maximising the size of the cache and minimising the number of reads from disk the overall response time for any operation is reduced.

Background Processes

There are a number of background processes belonging to each instance:

- **the database writer process** (`dbwr`) - writes modified blocks from the buffer cache to the database. The `dbwr` process of each instance writes blocks to disk when they are required by another instance.
- **the log writer process** (`lgwr`) - all Oracle instances share redo log files. The log writer allocates space in the redo log file keeping each instances redo log records separate, and then writes the entries to disk.
- **the checkpoint process** (`ckpt`) - this is an optional process that is responsible for updating control file and database file headers after a checkpoint. The log writer can also perform this operation; however, there are performance implications in using the log writer to do this.
- **the lock process** (lck0) - this uses the lock manager to prevent simultaneous changes to the same data by co-ordinating the buffer caches of all SGAs.
- **the system monitor process** (smon) - monitors and performs instance recovery for other processes. If an instance fails the smon of another instance rolls forward any committed work not written to disk or rolls back incomplete transactions and frees locked resources.
- **the process monitor process** (pmon) - performs process recovery when a user process fails. It cleans up the cache and frees any locked resources.
- **the archiver process** (arch) - archives redo log files on a per instance basis (Oracle Corporation, 1991).

7.2.4.2 An Oracle Database

The database can be defined as the physical disk space that stores the data and the logical structures containing data and consists of a number of files:

- control file
- redo log file
- init.ora files
- database file(s).

The control file contains information such as the database name, the database time-stamp and the names of database files and redo log files. It also holds the values of *init.ora* global cache parameters and other *init.ora* parameters that must be identical for all instances running concurrently.

Database files are where the data are stored. All instances have access to the same database files. The database files are stored on the parallel disk farm. In the configuration used here the data are striped across two of the three disks available to reduce the seek and fetch time for retrieving data blocks.

The *init.ora* file contains the parameters to create the SGA for each instance. Each instance has its own *initsid.ora* file where *sid* uniquely identifies the start-up file. Instances that only use public rollback segments (roll back segments generally available to any instance accessing the database) can share a single *initsid.ora* file. Each instance of Oracle on Hydra has an *initsid.ora* file. The instance name for this particular installation was Holly and the *sid* was derived from the node number, i.e. `holly0`.ora - `holly9`.ora.

Redo log files record all the changes that transactions make to the database data blocks providing a recovery mechanism in the case of system or disk failure. All instances sharing a database write to the same redo log file. If the file fills up then all instances must write to the next log file.

7.2.4.3 Oracle Server Processes

The Meiko Computing Surface is at a basic level a network of processors. Therefore, Oracle requires a number of server processes to handle communications and control between nodes.

There are a number of these processes.

- **The Database File Server** is dedicated to access the database files. The file server runs on a processor dedicated to file I/O only such as the MK050.
- **The Lock Server** provides currency control and synchronisation between instances and also run on dedicated processors.

- **The SCN (System Commit Number) Server** provides an incremental time-stamp for each transaction and is available to all instances. This server runs on a dedicated processor.

Finally, there is the pserver process which starts processes on behalf of local and remote requesters. The pserver requires a system identifier, *oracle_sid*. The *oracle_sid* uniquely identifies the logical host name of the Oracle instance required by a client.

7.2.4.4 Two -Task Architecture

Oracle is implemented on MCS using a 'two-task' architecture. This means that each application task starts a corresponding shadow task. The shadow task constitutes the Oracle kernel and is the only non-background process that has access to the SGA. The Oracle kernel code is shared which means it only needs to be loaded into memory once. Each application maps the same code segment and uses a private pointer to keep a track of its position in the program.

7.3 Arc/Info

The description of Arc/Info will be confined to the methods available within the application of linking to external databases, because these interfacing mechanisms form an important part of the research. There are three mechanisms for linking to external databases, which are the relate environment, cursor processing, and the Database Integrator which is a generic family of interface modules within Arc/Info.

The implementation of Arc/Info on the MCS is similar to other installations on a UNIX workstation, and therefore this section will concentrate on the data management side. Arc/Info v6.0 can manage spatial and attribute data internally using the INFO database, or externally through the relate environment or through the Database Integrator. The following describes each of the environments and the considerations that should be taken into account when using Arc/Info with external databases.

7.3.1 The INFO Database Management System

The INFO DBMS is a file based system that allows the user to read and edit the feature attribute tables and related INFO files (such as look-up tables). It should be noted that INFO can manage both data held directly in files within the INFO database, and external data. The Arc/Info attribute tables (PAT, NAT and AAT), boundary (BND) and TIC files are examples of the latter as they are not stored in the INFO database. As a result they are termed external files. They are in fact ordinary fixed format system files resident in the Arc/Info coverage directory. Each INFO database has a system directory containing a catalogue (file called ARCDR9), a default spool file (ARCNSP) and a number of special files. For each external file there are two INFO internal files, one of which contains the path-name to the external file containing the data, and the other of which defines the column names and format of the data (column width in bytes and data type such as Integer, Binary or Character). It is a simple matter to turn any fixed format file into an INFO external data file. Feature attribute tables must be external files referenced in an INFO database located in the coverage workspace. Related INFO files can be stored in any INFO database such as the coverage workspace or any remote INFO database.

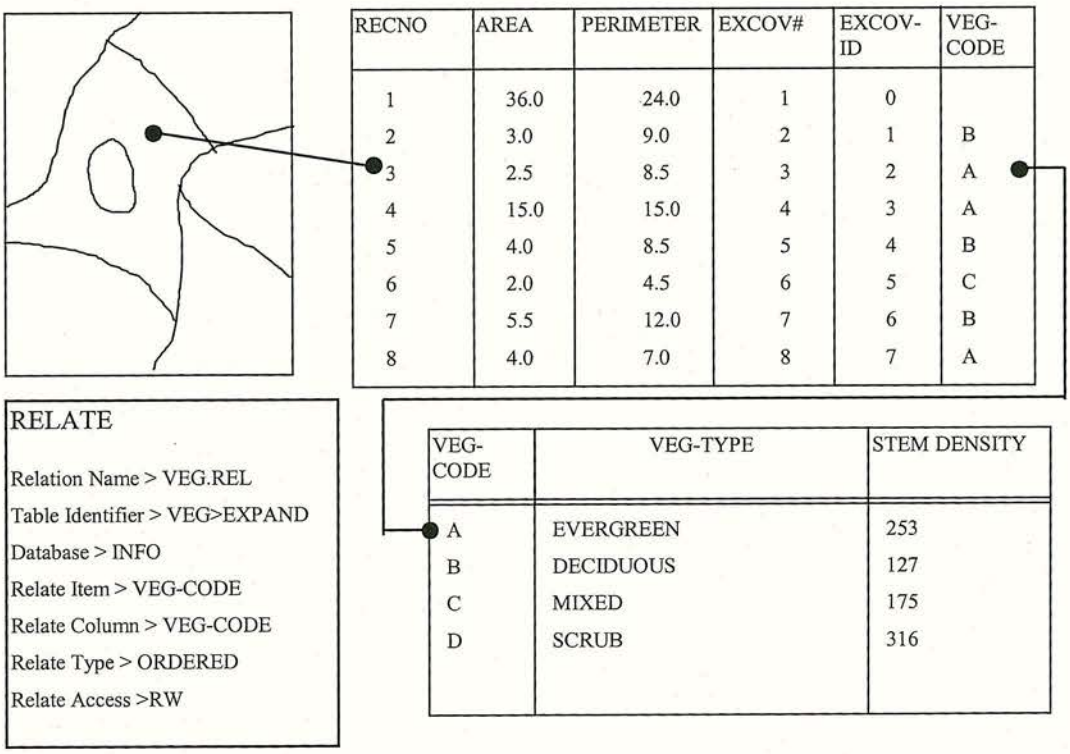
Files held in the INFO subdirectory are only accessible from the Arc/Info modules and the INFO database software, but external INFO data files may additionally be accessed by operating system commands (e.g. to sort the file on a column) or another user program. The basic contents of the INFO subdirectory have been described above, but the details are as follows: each INFO data file, whether internal or external is always associated with two disk files:

- ARC*nnn*DAT - This file contains either the data for the corresponding internal INFO data file, or the path-name to the external system file which contains the data.
- ARC*nnn*NIT - This file contains item definitions for reading the data (*nnn* refers to a three digit number assigned sequentially to INFO data files).

The ability of INFO to access external files means that many Arc/Info operations that manage or update the attribute database can take place without having to use the INFO DBMS system.

7.3.2 The Relate Environment

A relate in Arc/Info is a named relationship between an item appearing in a feature attribute table (or many tables) and either a related INFO file or a table in one of the supported DBMS. The relate environment exploits a relational database structure and could be described as a relational join across tables stored in multiple DBMS. It uses ESRI Info file processing libraries linked to routines containing embedded SQL for accessing the external DBMS. A record in a feature attribute table is related to a record in another table when their relate items match, or in other words a column in the feature attribute table forms the foreign key to the primary key of the related table, e.g. RECNO and VEG CODE (see Figure 7-5). Feature attribute tables, INFO files and external DBMS tables can be related to the feature attribute table of a coverage using a relate.



RECNO	AREA	PERIMETER	EXCOV#	EXCOV-ID	VEG-CODE
1	36.0	24.0	1	0	
2	3.0	9.0	2	1	B
3	2.5	8.5	3	2	A
4	15.0	15.0	4	3	A
5	4.0	8.5	5	4	B
6	2.0	4.5	6	5	C
7	5.5	12.0	7	6	B
8	4.0	7.0	8	7	A

RELATE

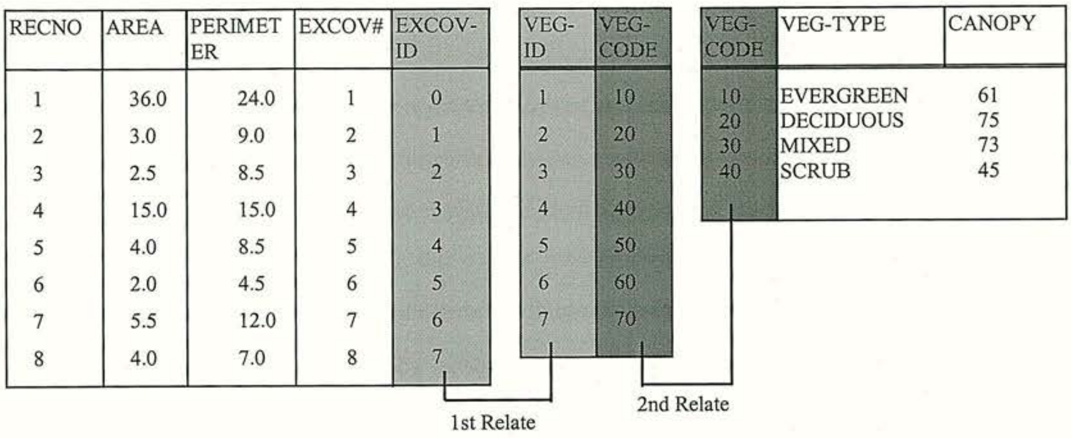
Relation Name > VEG.REL
Table Identifier > VEG>EXPAND
Database > INFO
Relate Item > VEG-CODE
Relate Column > VEG-CODE
Relate Type > ORDERED
Relate Access >RW

VEG-CODE	VEG-TYPE	STEM DENSITY
A	EVERGREEN	253
B	DECIDUOUS	127
C	MIXED	175
D	SCRUB	316

Figure 7-5:A Conceptual View of a Relate Environment (after ESRI, 1991)

Up to 100 different relates can be defined as part of the relate environment at any time, although only five can be used simultaneously during an operation. Normally relates will be lost on quitting from Arc/Info unless they have been saved. Relates are saved in a special INFO table, and can then be reloaded by using the name given the relate table when it was saved.

Arc/Info supports more than one level of relates, termed stacked relates, so that data can be accessed through other related files (see . Figure 7-6).



RECNO	AREA	PERIMETER	EXCOV#	EXCOV-ID
1	36.0	24.0	1	0
2	3.0	9.0	2	1
3	2.5	8.5	3	2
4	15.0	15.0	4	3
5	4.0	8.5	5	4
6	2.0	4.5	6	5
7	5.5	12.0	7	6
8	4.0	7.0	8	7

VEG-ID	VEG-CODE
1	10
2	20
3	30
4	40
5	50
6	60
7	70

VEG-CODE	VEG-TYPE	CANOPY
10	EVERGREEN	61
20	DECIDUOUS	75
30	MIXED	73
40	SCRUB	45

Figure 7-6: Stacked relates: an example of how three tables are related (after ESRI, 1991)

In other words one relate may be stacked inside another to allow three tables to be related together. It is not currently possible to have more than one level of stacking, so that a maximum of three tables can be related together.

Relates, once defined or restored from the INFO relate file, may be used at any point in the Arc/Info system where an INFO item (column) name could be used. Thus, for example, the relate SPECIES might define the relational join between a coverage PAT and a look-up table containing plant species names. To select polygons on the basis of a database attribute one would use the RESELECT command which has the following syntax:

```
RESELECT (<coverage>) POLYGONS (<INFO expression>)
```

in which the INFO expression would consist of an item (column) name, an operator such as '=', and a value. To use the relate, the item (column) name in the INFO expression is simply replaced by the relate name and the column to be used in the related table, thus:

```
RESELECT (<coverage>) POLYGONS SPECIES//(<column>) = (<value>)
```

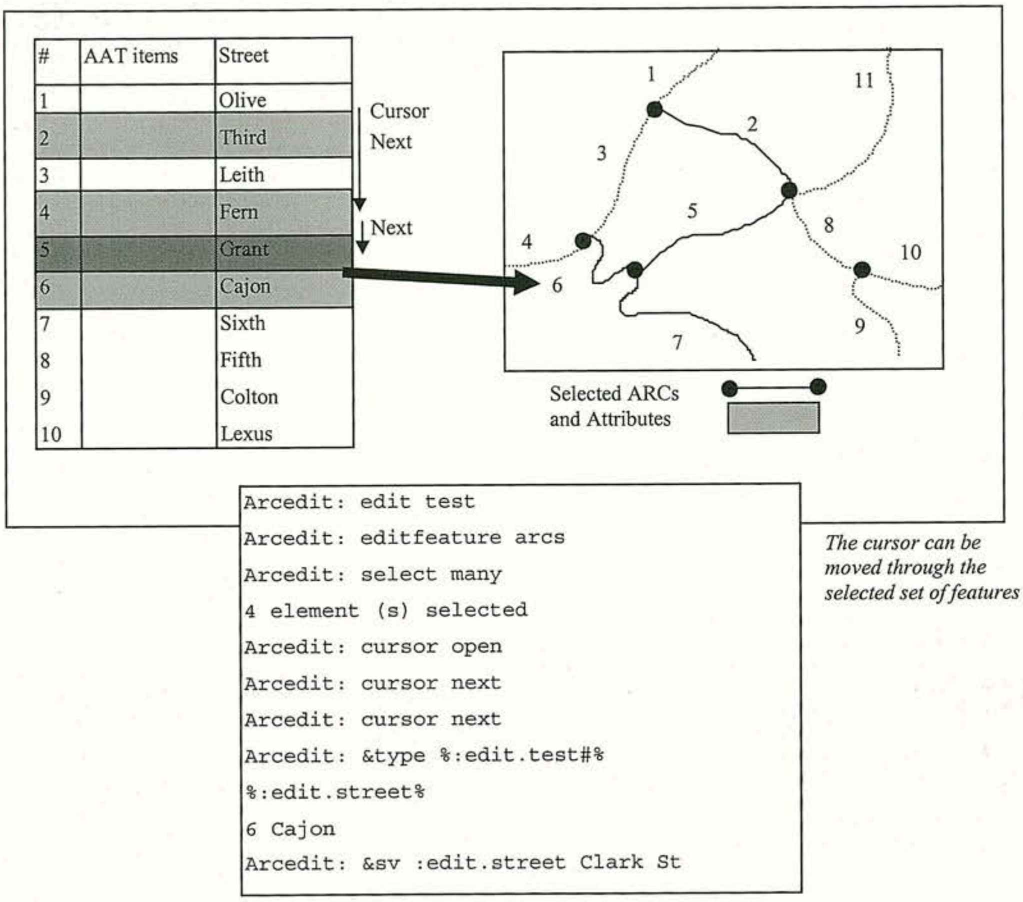
Note that the symbol "//" is used to separate the name of the relate and the name of the column from the related table, and that any column from the related table can be specified.

Relates support either one-to-one relationships or many-to-one relationships. An example of a one-to-one relationship is that between the PAT and an ORACLE table containing one row for each polygon. An example of a many-to-one relationship, is between the PAT and a look-up table containing the names corresponding to codes in the PAT. If, for example, a related ORACLE table contained multiple matching rows, then using the relate only the first to be encountered would be selected. In some circumstances it is acceptable to pre-process the related table or tables to overcome some of these limitations. For example, the restriction on stacked relates can be overcome in ORACLE by creating a *view* that establishes the required relationship between multiple ORACLE tables. The Arc/Info relate can then be made to the database view. Complex views, though, can considerably affect the performance of queries. This could be an area where parallel database environment could have a significant effect by, for example, using views based on parallelised relational joins. The one-to-many or many-to-many problem can sometimes be bypassed also by creating a database view. In this case the view would simplify the relationship to either one-to-one or many-to-one. If, for example, an ORACLE table contained details of planning applications, then a view could be created that was a count of the number of applications per polygon, and which thus had only one row per polygon.

7.3.3 Cursor Processing

Relationships which are either one-to-many or many-to-many are difficult to handle effectively using the relate mechanism, as was described above. It is now possible to overcome these limitations by means of defining and using a cursor. A cursor is not a physical entity in this context but a programming concept. Cursors are a mechanism for accessing a selected set of coverage feature attributes, INFO file records, or DBMS table rows, one element at a time. Cursor is a DBMS industry-standard term for managing a pointer to a particular row in a database table

Cursors support one-to-many relationships between INFO files or external DBMS tables, allowing one to step through each of the many records or rows related to the current row as can be seen in Figure 7-7.



#	AAT items	Street
1		Olive
2		Third
3		Leith
4		Fern
5		Grant
6		Cajon
7		Sixth
8		Fifth
9		Colton
10		Lexus

```
Arcedit: edit test
Arcedit: editfeature arcs
Arcedit: select many
4 element (s) selected
Arcedit: cursor open
Arcedit: cursor next
Arcedit: cursor next
Arcedit: &type %:edit.test#%
%:edit.street%
6 Cajon
Arcedit: &sv :edit.street Clark St
```

The cursor can be moved through the selected set of features

Figure 7-7: A schematic example of a cursor being used to access a selected set of arcs in ArcEdit (after (ESRI, 1991)

Both Arc/Info and the Arc Macro Language (AML) support cursors at Revision 6.0. A cursor may be declared at any point in either the ArcEdit or ArcPlot modules and thereafter the results of a database query displayed as a pop-up window. Alternatively AML directives and functions can be used to extract data from the cursor, using the row and column numbers, so that it can be used as input for other tasks. In Arc/Info, cursors are designed primarily as a mechanism for displaying and updating INFO files and DBMS tables using the graphic user interface which is OPEN LOOK or MOTIF based.

7.3.4 The Database Integrator

The Database Integrator is a generic family of Arc/Info software interface modules. Each of the modules provides access to a single DBMS including INFO. However, multiple DATABASE INTEGRATOR modules can be used concurrently to issue relates to different external databases (Figure 7-8).

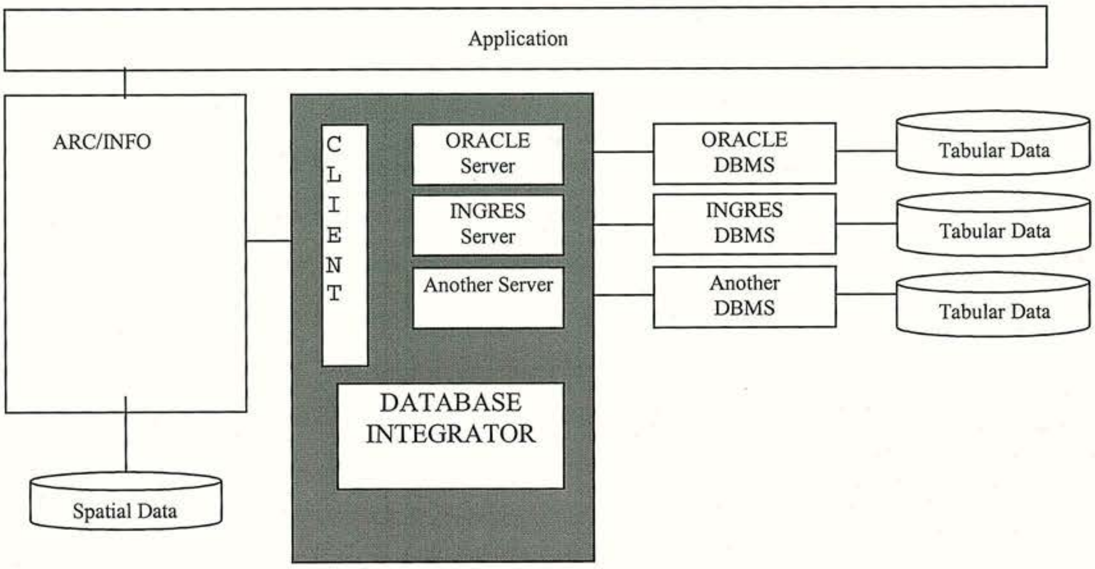


Figure 7-8: An illustration of the relationships between the data, the software modules and the software systems (after (ESRI,1991))

A database in Arc/Info terms is a logical concept for a group of tables stored in a DBMS to which the user has access. A DBMS can manage many databases.

The DATABASE INTEGRATOR does not require the INFO DBMS for any of its functionality and will run even if INFO is removed from the Arc/Info configuration (although it does still require the Info file structure to remain) because ESRI has duplicated all of the necessary code!

The DATABASE INTEGRATOR uses a CONNECT command to login to and maintain a communication link between Arc/Info and the DBMS. The database definition file contains the parameters required to connect to specific DBMSs and will allow the user to connect to a specific instance (in ORACLE) of the DBMS. However, it is not possible to connect to the same database using the same database definition file more than once in the same Arc/Info session. In each Arc/Info session a user may have up to five simultaneous database connections. Each connection creates its own process that maintains the communication path between Arc/Info and the DBMS. Data can be transferred/converted from INFO to the DBMS and vice versa.

The DATABASE INTEGRATOR permits the use of native mode SQL queries submitted to external databases. The use of native mode SQL is of great importance because it submits a single query to the DBMS, unlike the relate environment where a query is submitted for every selected record in the INFO data file and should have a considerable impact on performance. This is another area where a parallel database could enhance performance further by parallelising the SQL code to reduce the time for external DBMS retrieval.

7.3.5 Special Considerations in the Use of ORACLE

In all situations where the topology is altered and rebuilt in Arc/Info, the software will create and/or update the PAT, NAT, and AAT. These files will thus always maintain a one-to-one relationship with the graphic features and attributes such as length and area will remain correct (i.e. there will be one row in the PAT for each polygon in the coverage).

The situation becomes more complex where some of the data are held outside the Arc/Info system, for example with some attribute data being held in ORACLE. In these circumstances the PAT, NAT or AAT might only hold the default attributes for each feature (i.e. the internal feature number - (<coverage>)# - the user ID number - (<coverage>)-ID - and attributes such as length, area and perimeter), with all of the user defined data held in a related ORACLE table. The ORACLE table would be related to the feature attribute table by means of the (<coverage>)-ID number. Once this system has been established, a relate defined and the ORACLE table populated, the system can function as if the ORACLE data were seamlessly appended to every row in the feature attribute table. It does, however, need to be noted that the Arc/Info software is not itself aware that the external database table exists. This means that when the coverage changes, for example by the addition of a new polygon, the topology can be rebuilt which will update the feature attribute table, in this example by adding a record for the new polygon. No changes, though, will be made to the related ORACLE table. It is left to the user to update the external database, by adding a new record containing the id number of the new feature. Similarly, deleting a feature would mean that the ORACLE table would contain a row that no longer matched to any of the coverage features, and the user would decide whether or not to remove it.

7.4 Smallworld

Smallworld GIS is an object-oriented system which was mainly used in this research to digitise the gas mains network which formed a central part of the large corporate database developed in Phase II of the performance testing. The Smallworld system is quite complex because it is an object-oriented environment built on top of a relational database. The following section describes the database, known as the Version Managed Data Store (VMDS), how it is managed and updated, and finally how it links to the Oracle RDBMS.

7.4.1 VMDS (Version Managed Data Store)

The Version Managed Data Store (VMDS) is the name given to Smallworld's own internal database. It is based on the tabular model that underlies other commercial relational database applications.

The database comprises a set of tables, and each table contains a set of records. A database record in Smallworld behaves much like any traditional slotted object, where the record has *fields* that correspond to slots in an object. All records in a table are the same *shape* - like a collection which contains objects which are of the same class.

One or more of the fields in a record forms its primary key. The value of this set of fields must be unique to each record in a particular table. The primary key effectively gives the record its identity - two records are considered the same if they come from the same table and have same value of primary key.

A database table is implemented in Magik[7] (Smallworld, 1991a) to behave like a form of collection. Therefore, database collection objects respond to methods such as *.size*, *.an_element()* and *.elements*(). The standard database collection may be sub-classed to provide specific behaviour for particular tables, if required.

[7] The Smallworld object-oriented programming language

Each element of a collection (i.e. a record) is an object and all the elements of a particular table have the same class. They have methods for accessing each field (in the same way that a slotted object has slot access methods). They also have data dictionary behaviour, with various methods returning information about the structure of the record. The exemplar [8] for the class can be generated automatically, so that any database may be opened and be immediately capable of interrogation.

7.4.1.1 Field Assignments

Field assignments differ between the VMDS and foreign databases. In the case of the VMDS, field assignment works directly on the database, so the database is updated as the field values are changed. In the case of foreign databases such as ORACLE, the record object is treated in Smallworld as a true slotted object. Therefore, updating its fields will have no effect on the data stored in the underlying database. The collection must be explicitly committed with the new record value (Smallworld, 1991c).

The primary key value gives the record its identity; therefore, it is not permissible to change the value of a primary key field of a VMDS record object.

7.4.1.2 Database Views

VMDS collections are grouped together in a hash table in an object of class *ds_view*. The hash table is in a slot named *collections*, and a collection is referred to by its name expressed as a symbol. There may be a number of *ds_views*, each of which contains one or more files. A particular file may be in more than one view at a time. The primary purpose of a view is to group collections which are to be committed or rolled back together (ensuring atomic commit), and to enable different versions of the same set of files to be accessed at one time. Each *ds_view* behaves like a separate user, and must conform to the same concurrency constraints.

[8] An exemplar is an instance of a particular class that is in some way typical or holds default values. New class instances can be created by sending a message to the exemplar which will make a copy of itself and perform any special initialisation that is required.

7.4.1.3 Versioning

Traditional database systems are based on the premise that there is only one consistent state of the database at any one time. Two users may seemingly change the database at the same time, but they are prevented from changing the same or related records simultaneously by locking or forced rollback.

The VMDS takes a different approach. It is designed on an optimistic principle that in practice most changes that different users make concurrently do not conflict; and that conflict, when it does occur, can usually be resolved by the operator selecting which parts of the two updated versions should be merged to make one version. The VMDS is not suited to applications where a single up-to-date state is mandatory at all times. The VMDS is designed for efficient storage and access to multiple versions of the same data. It does this by keeping disk blocks which differ, rather than by cloning data en masse.

Before the VMDS can be used, it must be initialised. This is true for both open and closed images. The following protocol will perform the required operation:

ds_environment.init(gather params)

Where params is a list of key, value pairs and *ds_environment* is a global object that contains information about the VMDS as a whole.

7.4.1.4 Access to an ORACLE Database

Smallworld allows access to the ORACLE database through the Magik language. An ORACLE table or collection of tables is treated in a similar fashion to tables held internally in Smallworld and can be manipulated using methods written in Magik.

To access the ORACLE database (only a single database can be accessed), the following protocols should be used: **oracle.open** *(login_name, password)*. This returns an *oracle_user*. The ORACLE object is a single global which corresponds to *ds_environment* for the VMDS (Version Managed Data Store). The *oracle_user* is an object connected to the ORACLE database. There may be more than one in one Magik Environment. An *oracle_user* crudely corresponds to a *ds_view*, and is the unit of commit and rollback.

All the methods described in the next section can be applied to an *oracle_user*:

- **open_collection** (*name*), which returns an *oracle_collection*
- **commit***()*
- **rollback***()*
- **close***()*

Tables are not opened automatically (as they are in VMDS), but must be explicitly opened.

Changes to ORACLE collections can be committed using *commit()* or undone using *rollback()*.

Smallworld was briefly linked up to the Oracle RDBMS, but due to the very slow elapsed times measured during test runs it was decided that this was not a viable route to follow.

7.5 Summary

The test environment was constructed from a number of elements. The hardware platform was a Meiko Computing Surface (MCS), consisting of:

- 10 SPARC nodes - each with at least one disk attached to it
- 24 transputer nodes
- a parallel disk farm dedicated to Oracle.

All of the nodes were connected together using a very fast interconnect called the CSN.

There were two main software platforms, Oracle v6.2, a multi-user version of a relational database management system, and Arc/Info v6.0. The Oracle database was a parallel version designed to utilise the whole of the MCS. Oracle processing took place at two different levels: on the SPARC nodes, and on the transputers. The SPARC nodes were responsible for setting up and running individual Oracle instances, while the transputers were used to run all of the background processes, including the parallel lock managers which controlled the updating of individual data blocks. A third element of interest was the parallel disk farm because it had its own buffer cache which could have an effect on the performance test results.

The Arc/Info environment was established on individual nodes of the MCS. The main feature of interest in Arc/Info were the interfaces available through the relate system, and the Database Integrator to external databases, and in particular the Oracle database. Arc/Info was set up and a number of relates successfully established.

Smallworld was also established on individual nodes of the MCS. An interface mechanism existed between Smallworld and Oracle through a series of methods supplied by Smallworld. A connection between Smallworld and Oracle v6.2 was established and data retrieved from the Oracle database. However, the performance of the interface was very slow and it was cumbersome to use. These two aspects of the Smallworld interface forced the software to be discounted for performance test purposes. The Smallworld GIS software was used extensively in the design and creation of the large corporate database used in phase II of the testing procedure with great success, in particular the ability to digitise from raster images from the screen proved very useful.

Once the testing environment was established it was necessary to create the databases for phase I and phase II. The testing was split into two phases to allow a series of baseline tests to be measured using an artificial dataset, and a number of more realistic tests using real data. The artificial dataset would make the tests easy to control and calibrate, to allow an accurate picture and understanding of the database and GIS interface mechanisms in a parallel environment. The real data would give a truer picture of the performance of the GIS parallel environment. Details of the two databases can be found in chapters eight and nine.

8. Phase I - Pilot Database

As mentioned in the previous chapter, the performance testing process, was split into two phases: known as phase I and phase II. This was done for several reasons, firstly, to provide an environment where unfamiliar hardware and software could be brought together for trial purposes, secondly, to aid development of a test environment and performance gathering techniques, and finally to allow for the development of two different approaches to testing. The first used artificial data to test the parallel database environment under controlled conditions, and the second used real data from British Gas (Scotland) to test the parallel database environment under more realistic conditions. One of the major criticisms of benchmarks is that they use artificially created datasets that are not representative of the corporate databases they are mimicking, and are often too small to show up difficulties relating to I/O and memory bottlenecks. It was hoped by taking this two stage approach some of these criticisms could be overcome.

The following two chapters describe the development of the database environments created for phase I and phase II. This chapter concentrates on the pilot database developed for initial testing of the environment. The tables and queries used in this phase were developed for a research project examining the performance testing of the use of corporate GIS with distributed databases (Chan, 1993) and, due to the nature of the multi-user Oracle database, were found suitable for testing a number of aspects of this parallel/GIS database environment. The testing suite is divided into six stages and provides data and queries for testing the performance of relational join strategies, the effect of indexing on elapsed time and the effect of the buffer cache on data retrieval times. The use that this project makes of the tables and queries designed by Chan is unique to this research project.

8.1 Background

The use of commodity components, such as SPARC processors, in parallel architectures has made the definition of what constitutes a parallel computer distinctly unclear. A cluster of SPARC processor based workstations networked together and used in parallel to solve corporate data processing difficulties, could technically be classed as a parallel computer, and has often been described as *loosely coupled* parallel architecture. Similarly, the distinctions between distributed Oracle databases and multi-user Oracle (v6.2) on the MCS have also become rather hazy. One of the main differences is the physical distance between the processors. For the MCS they lie a few inches apart and are linked together using a fast, dedicated interconnect, while in a conventional distributed database system they can be hundreds or thousands of miles apart, connected via ethernet and optical cable. However, the methods of addressing different nodes in the system, and the structure of SQL queries for retrieving data from multiple nodes are very similar. Therefore, the database and SQL query suite, developed for investigating the use of corporate GIS based on distributed database technology, were found to be both pertinent and applicable for the initial testing of the parallel database environment. The obvious difference is that instead of accessing data from multiple databases through a distributed system, the parallel database would provide access to the same database for all nodes available on the MCS.

The research investigated the effect of data distribution on the performance of SQL queries of both textual and spatially-related data across a network, and examined the implications for architecture design on data distribution and query performance. A series of database tables and SQL queries were developed to analyse the effects of distributed-databases, query construction and GIS models on the performance of hardware, software and communication infra-structure. The author was involved in this work in an advisory capacity, and gas mains data from this project which examined the use of parallel databases with corporate GIS, was donated to form part of the distributed database.

8.1.1 Aim of the Distributed Database Research

The aim of this part of research project was to simulate a corporate GIS environment where data were located in a distributed database system environment (DDBS) to determine appropriate strategies for data distribution and the optimisation of query processing. A series of performance tests were developed to investigate the effect of different permutations of data distribution on a single computer and a series of computers connected via a network. Data in topological, vector, raster and attribute formats including OS data of Cumbernauld, gas mains data from British Gas (full description in Phase II), street works data and customer data were combined to create a simplified corporate database. The individual databases were located on VMS and UNIX systems (including Hydra) based at different locations around the University of Edinburgh and connected via LAN (ethernet) and FDDI communication links (Figure 8-1). Oracle v6.2 and ARC/INFO 5.1 were employed, being commonly used database and GIS applications.

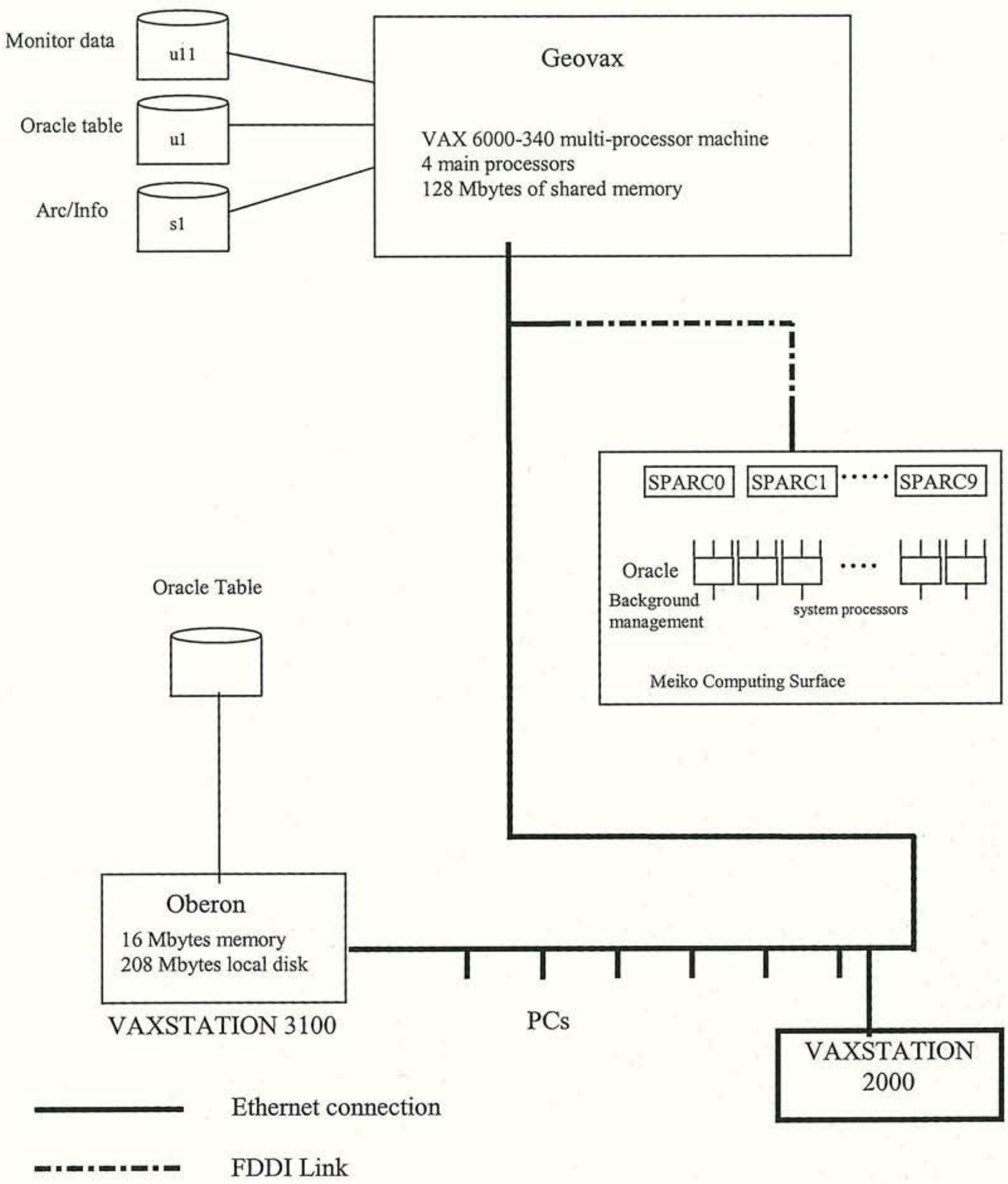


Figure 8-1: The hardware platform for Distributed Database Performance Tests (after Chan, 1993)

The research into distributed database performance used three servers available over the University of Edinburgh campus. Two of the machines, Geovax and Oberon, were based in the Department of Geography while Hydra was based several miles away and accessible through the University network using a mixture of ethernet and optical fibre (FDDI) links.

Geovax was a VAX 6000-340 and Oberon a VAX3100. A full description of Hydra can be found in chapter seven.

The research should be of interest to corporate GIS users for a number of reasons. Many large organisations are located over a number of sites, often in different parts of the country or in different countries and need to have corporate data accessible to all sites to gain a full business picture. Similarly the corporate GIS will also have to span many sites. It is important to know the performance implications involved in manipulating spatial data from several different sites, particularly as it involves the use of communication links shared by the whole organisation. Strategies for retrieving data that minimise network bottlenecks improve the performance for the whole organisation and reduce processing time for GIS functions. These strategies could be critical if deadlines are very tight.

The research also has an impact upon corporate data integration strategies involving GIS, whether data are located locally or remotely. Corporate GIS usually require access to data from many different locations. By considering the location of the data, the type of database, and the processing power of the server, it is possible to develop an efficient integration strategy.

8.1.2 The Distributed Database Performance Test Design

To investigate the behaviour of the distributed database environment under different operating environments, data loadings and optimisations, six stages of test programme were designed. The first group of tests was conducted on individual nodes. Their purpose was to provide a series of control results for comparison with results collected from the second group of tests, which were conducted using multiple database nodes in a distributed environment. A full description of the project can be found in Chan (1993).

8.1.2.1 Single Node Queries

The first group of tests was split into three stages and used to investigate the behaviour of single nodes under a number of different circumstances.

Stage I - Single Join on a single node

Queries of single relational joins on tables of different sizes were run on individual nodes of Geovax, Oberon and Hydra to examine any threshold effects or abrupt changes of behaviour in the database query performance. The query results returned a fixed number of rows of data based on the size of the data table to allow for comparison of systems under different conditions of caching, attribute indexing, and data load (see Table 8-1).

Table 8-1: Example of Indexing Effect on Elapsed Time Using a Fixed Number of Returned Rows (after Chan, 1993)

No. of rows returned	Index Strategy 1 Elapsed time (secs)	Index Strategy 2 Elapsed time (secs)
6667	81.55	117.18
3333	37.5	57.27
2778	36.28	44.22
2222	26.53	37.13
1667	18.26	26.85
1111	13.61	18.98
555	6.94	10.3
333	2.78	4.66
222	2.81	3.34
111	1.11	2.26

Stage II - Multiple Joins on a Single Node

This phase of testing had a two-fold purpose - first to examine the significance of how tables were ordered in a relational join and, secondly, to identify the optimum table order for relational joins used in test *stages III* and *V*. A number of multi-table multi-join queries were carried out using four tables with three relational joins. The four tables were indexed on their foreign keys. The queries were designed to retrieve data sets of 850 tuples, 1700 tuples and 3400 tuples respectively.

Stage III - Spatially-related Queries on a Single Machine

The third phase investigated two different database model approaches in GIS - integrated DBMS and hybrid DBMS, retrieving spatial and corporate information from a single machine using ARC/INFO and Oracle. A view was constructed in Oracle using a query with optimal relational join ordering (developed in *Phase II*) and used in conjunction with other tables and files to simulate corporate applications that generate complex queries over multiple tables. To simulate an integrated approach, a PAT file from ARC/INFO was converted to an Oracle table and queries run against this table and the view table (Figure 8-2). To simulate a hybrid system a link was set up in ARC/INFO to relate the spatial data to the view table held in Oracle. Queries were generated selecting and re-selecting spatial data using ARCPLOT.

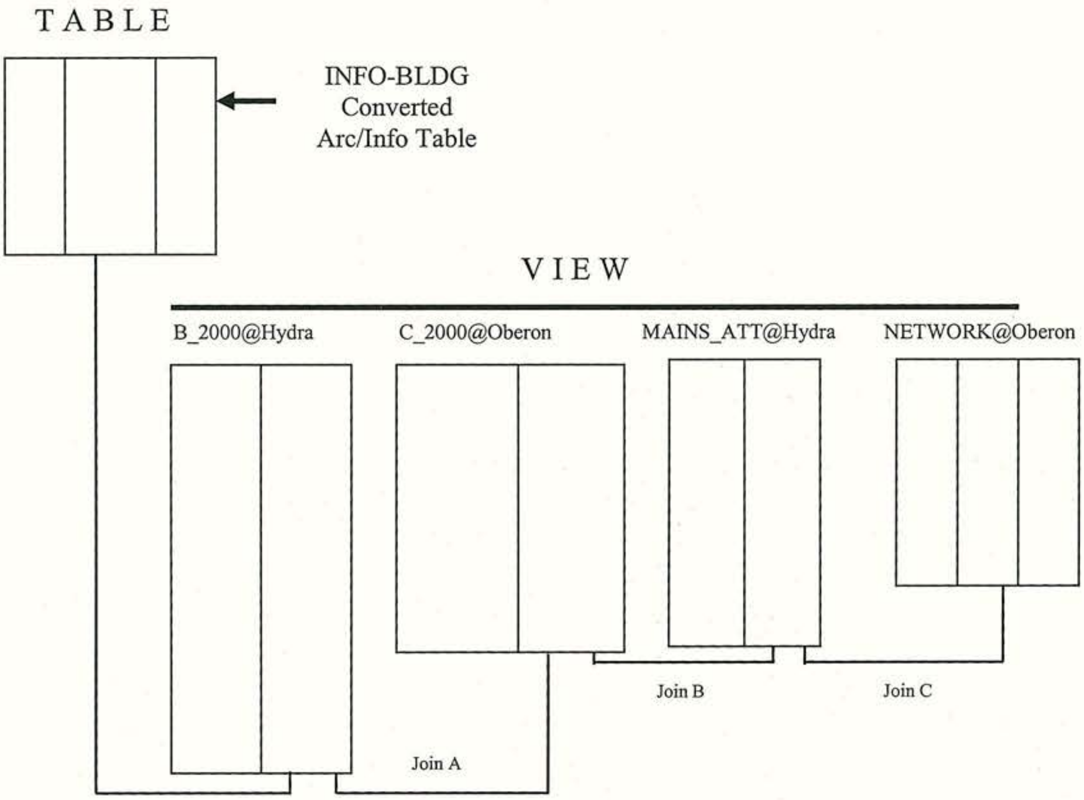


Figure 8-2: Diagram to Show the Link Between a Spatial Table and an Oracle View (after Chan, 1993)

A view of the data was constructed using queries developed in stage II, to generate a series of complex queries over multiple database tables. The two GIS models were approached in the following manner. Both models used the same data: the integrated approach had both the spatial and attribute data held in Oracle; while the hybrid approach relied on Arc/Info to manage the spatial data and Oracle the attribute data. Queries for both GIS models were generated by selecting and reselecting spatial entities from the screen.

The three stages outlined above provided a series of ready made SQL queries for performance tests for use with individual nodes on Hydra. The queries from the stages were taken and used for testing the behaviour of single nodes on Hydra, to compare response times for each node against the performance of the other nodes in the parallel environment, and to locate any performance black spots. Once the nodes were characterised individually, a strategy was then developed for using nodes in parallel, and identifying roles for each of the nodes. These results were used as a base line set of performance indicators for comparison with performance statistics gathered from other test runs.

8.1.2.2 Multiple Node Queries

The second three stages of parallel testing concentrated on the use of relational joins to access multiple tables held both locally and remotely in Oracle databases. The tests aimed to examine the effect of data loading when larger tables were accessed in a distributed system to try and derive a set of rules for table location. The final tests investigated the effect of two GIS approaches i.e. the hybrid and integrated GIS models, on distributed databases.

Stage IV - Distributed Single Joins

The aim of this phase of testing was to investigate the effect of data distribution on single join queries across a network using different data loading and different percentages of data retrieved. Databases were located on three computers Geovax, Oberon, and Hydra.

The connections between Geovax and Oberon consisted of local ethernet links, while the connection between Oberon and Hydra was between remote sites and used an optical fibre (FDDI) with ethernet at either end (see Figure 8-1). Oberon was used as the local site and Geovax and Hydra were remote sites. The processing power of the three computers ranged from Hydra lying at the top end of the processing scale to Geovax at the bottom. Thus, it was possible using this hardware configuration to define both local sites that had more processing power than their remote counterparts, or vice versa. The results of the different permutations for siting tables were compared with each other and with the results collected in Phase I.

Stage V - Distributed Multiple Joins

This part of the testing investigated the effect of data distribution on queries utilising multiple relational joins in a distributed database. The intention was to begin deriving some rules of thumb for optimising query performance by the pattern of data distribution. A number of join queries were performed between a table and the view constructed in *Stage II* (Figure 8-2). All foreign keys were indexed. Sixteen permutations were formed using four tables stored at two different sites. Each site was tested as both a local and remote site.

Stage VI - Distributed Spatially-Related Queries

The final phase of testing investigated the performance of different distributed DBMS approaches on two different models of GIS. Using an approach similar to that in *Stage III*, tests were carried out on integrated and hybrid GIS systems. Arc/Info was used as an example of a hybrid GIS, with spatial data stored directly in the GIS and attribute data stored in Oracle. Arc/Info was also used to simulate an integrated GIS. Both spatial and attribute data were stored in Oracle and accessed through the Database Integrator.

These three test stages provided a series of queries suitable for testing the performance of the database with an increasing workload. The workload started with one *user* on a single node, submitting a query with a single relational join. The workload was built up to ten users on all available nodes of Hydra, simultaneously submitting queries containing three or four relational joins.

One of the intentions of the testing procedure was to discover whether there were any permutations of users and workload that were more successful than others. For example, what was the effect of all *users* submitting queries from a single node, compared to users being spread out over all of the available nodes. This was of particular interest if the nodes on the machine were to be partitioned for different uses, or user groups, especially when submitting long, complex transactions.

8.2 Description of the Pilot Database

The pilot database consisted of a number of tables of artificial data, developed purely for database testing purposes. There were five main tables: a gas mains table; a gas mains attribute table; a buildings table; a customer table; and finally a polygon attribute table from Arc/Info containing co-ordinate data for individual gas mains. The gas mains table contained attribute data for each individual gas main in the network. This table was originally developed for queries testing the performance effects of distributed databases on corporate GIS. The table, along with the polygon attribute table and the mains network table, was used in Phase I testing (chapter eleven) to establish that all of the different database interfaces available from within Arc/Info were connecting to the Oracle database successfully and that data retrieval was satisfactory.

8.2.1 The Pilot Database Model

The database model was a very simple one (see Figure 8-3) but provided enough flexibility to begin initial tests on the parallel database system which could provide an understanding of the workings of the different elements, check the functioning of Oracle on individual nodes, and record some base performance statistics.

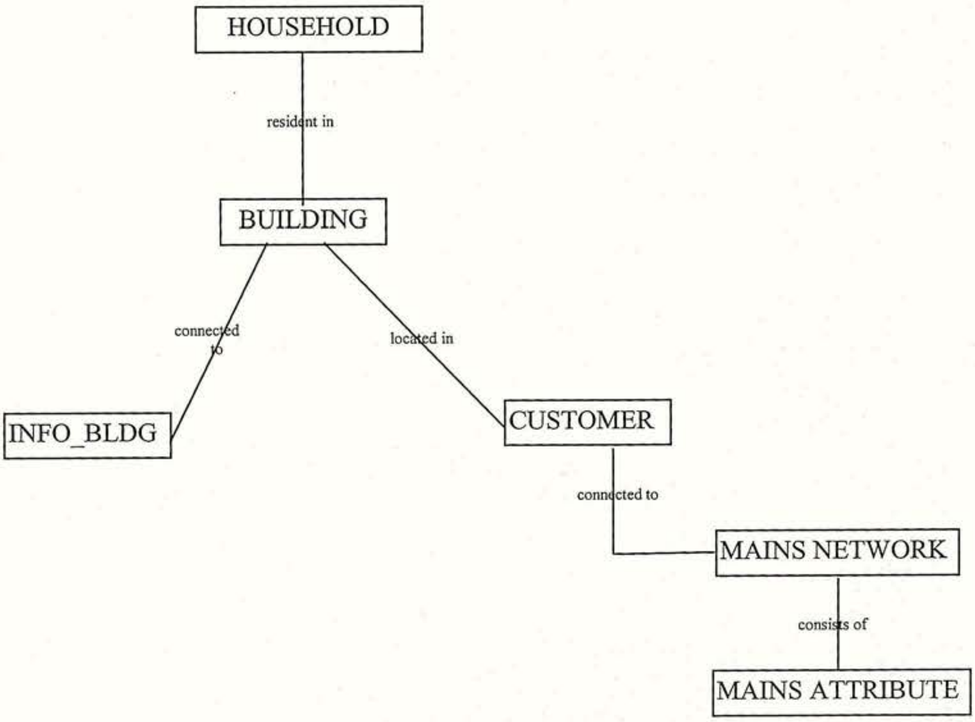


Figure 8-3: Pilot Database Data Model

The five entities shown in Figure 8-3 have the following attributes (see Table 8-2):

Table 8-2: Attributes for Pilot Model Entities

Entity	Attribute
Customer	customer#
	building#
	name
	address
	town
	postcode
	pipe network#
Building	building#
	description
	category
	building use#
	label point#
Mains Attribute	mains#
	network#
	siphon#
	injection#
	governor#
Mains Network	name
	network#
	length
Info Bldg	x
	y
	bldg id
	bldg#

The buildings and customer tables were the main driving tables for performance testing and were used extensively in most of the tests. A number of versions of the two tables were created containing differing numbers of rows (see Table 8-3) to simulate different data loadings. Indexes were also built on the tables to investigate the effects of a number of indexing strategies.

Table 8-3: List of Tables for the Pilot Database (after Chan, 1993)

Table Name	Type	Number of Rows	Number of Attributes
B_60000	Building	60000	5
B_30000	Building	30000	5
B_25000	Building	25000	5
B_20000	Building	20000	5
B_15000	Building	15000	5
B_10000	Building	10000	5
B_5000	Building	5000	5
B_3000	Building	3000	5
B_2000	Building	2000	5
B_1000	Building	1000	5
C_20000	Customer	20000	8
C_10000	Customer	10000	8
C_8333	Customer	8333	8
C_6667	Customer	6667	8
C_5000	Customer	5000	8
C_3333	Customer	3333	8
C_1000	Customer	1000	8
C_667	Customer	667	8
C_333	Customer	333	8
MAINS_MINE	Mains Attributes	1000	5
MAINS ATTRIBUTE	Mains Network	20000	3
INFO_BLDG	Mains co-ordinates	14567	4

The SQL suite consists of some fifty queries developed for use in the performance test stages described above. A list of SQL queries used in phase I can be found in Appendix A.

9. Phase II - Corporate Database Design

The purpose of this chapter is to present the different stages that together provide the basis for system testing in Phase II. The work falls into a series of sections, from the designing and building of an organisational datamodel through to the production of a series of data retrieval queries. The results of these stages provide the testing environment for phase II and consist of two GIS applications holding gas mains data, a parallel database containing approximately two years worth of organisational data, and a series of GIS and database queries to run on the parallel system. These stages, although a laborious and lengthy process to go through, were necessary to the research to provide a realistic workload for the parallel platform to manage. Using *real* data, with all its quirks and difficulties, reduces the artificial nature of the testing procedure as communication weaknesses, bottlenecks and gremlins encountered using large volume, working data with several applications linked together can be expected to manifest themselves.

The large corporate database held in Oracle was designed to look like a real utility problem. The underlying gas mains networks were digitised from working maps taken directly from the paper map library belonging to British Gas (Scotland). The digitising exercise coincided with the huge Digital Records project undertaken by British Gas (Ives, 1993), to turn their paper map-base into a nation-wide GIS. This gave a much clearer insight into the workings of a gas utility company and access to internal reports and documents improved the understanding further. The gas utility database designed for this project does not reflect the full complexity of a corporate database for such a large organisation, but has brought together the essential elements to allow for a much more thorough testing of the parallel database environment for use with corporate GIS. The size and scope of the database also made provision for both GIS and transaction processing type queries to be run either separately or together.

There are a number of problems associated using *real* data for testing purposes. It is difficult to design and calibrate tests, particularly those requiring precise numbers of rows returned, or those using selectivity factors to retrieve percentages of data, because it is difficult to guarantee the content or level of data returned by a query (DeWitt, 1993). This becomes even more complex when relational joins are involved. Therefore, it was important that results from phase I were available to use as a base-line for comparison with results from phase II.

The layout of the chapter is as follows. The first two sections describe the design process of the datamodel using Entity-Relationship modelling to create a series of inter-related entities representing the base building blocks of various organisational roles and processes. The datamodel is built upon a fictional gas utility company, Gas 'R' Us ltd., and based on information provided by British Gas (Scotland). The main focus of the utility company, and therefore the model, is the gas mains network because this is the fundamental function of the business and seen as the main target for GIS. However, emphasis is also placed on both the customers and their billing system and other areas of the business which could be integrated using GIS. The result of the design process was a model of some 37 entities with defined relationships which was translated into SQL code and loaded into the Oracle database.

The next two sections of the chapter records the physical construction of the corporate database and the process of populating it with a significant quantity of data, with the final database containing approximately 15 million rows of data. The database held details on a customer base of approximately 1 million, which was thought to be a reasonable representation for a gas utility company based in Scotland. The total population of Scotland stands at a little over 5 million, and much of rural Scotland does not have access to piped gas services.

9.1 The Data Model Background

9.1.1 Introduction

This section describes the design of a data model based on a fictitious Utility company, known as Gas 'R' Us ltd, the results of which will be converted to an Oracle database used to performance test the parallel database and GIS applications loaded on the MCS (Meiko Computing Surface).

There are three design stages described in this section. The first consists of a description of the type of organisation the model is based upon. The second stage describes the software application used to create the model, dipping briefly into entity-relationship modelling and its terminology and notation. The third stage consists of an in-depth description of all parts of the data model.

The main aim of the data model is to provide an environment for testing the performance of the parallel system, the parallel database, the GIS applications and external databases, incorporating some of the philosophy and methods devised in standard benchmarking tests, that have relevance to the project.

The function of the data model and database is to create an information infra-structure for a gas company that has both complexity and enough dimension to provide a fair simulation of the data requirements of such a company. It is important that the results of the subsequent tests have a reasonable level of relevance to the business environment to provide an insight into the performance and usability of parallel database systems with GIS.

9.1.2 Model Criteria

When building the model there were a series of five conditions that the model was required to meet and which are now described:

9.1.2.1 Database Size

There was a need to produce a test environment of a realistic size. Results from tests conducted using small datasets are very often misleading because the data will usually fit neatly into memory and are therefore unlikely to highlight potential weaknesses of the system. If a parallel GIS system is to be adopted by business (either public or private) it will be required to manipulate millions of rows of data regularly in a robust and efficient manner. Therefore it is necessary to show the performance at the levels of usage envisaged by the end users.

9.1.2.2 Database Complexity

Organisational data are often very complex; for example, a simple gas main can have between 15 and 20 attributes associated with it, detailing, size, material, pressure etc. It is important to try to capture some of this data complexity in the datamodel as it will have an obvious effect on the performance of the system. The complexity of stored objects will affect data storage, data retrieval times, and speed of communications between the GIS and database, and consequently user response times.

9.1.2.3 Support bed for testing

There is a need for many different types of test in the research - analysis of the performance of serial GIS in a parallel environment, the performance of the parallel database, the communication between GIS and database, the performance of the system under different work loads, and levels of user are a few examples. Therefore, the model must contain the different types of table and data necessary to do this. This will include a mixture of lookup tables, tables with many columns and tables suitable for transaction processing.

9.1.2.4 Multiple Users

The projected increase of users for any given application, due to increased use of both the Internet and intranets, necessitates that the environment will provide support for multiple users. These users will generate a mixture of short transactions resulting from transaction processing type queries and long transactions resulting from complex queries. The performance of the system under different user loadings should be analysed.

9.1.2.5 SQL Links

The environment must provide support to simulate both integrated and hybrid GIS datamodels, and for GIS linking to the database using off the shelf GIS packages. Comparison of the performance of GIS applications using external databases as the main data repository against their standard one will give an indication of the viability of using a parallel database as part of an integrated GIS model.

9.1.3 Data Model Background

The data model designed for this research is based around the British Gas Digital Records strategy (Ives, 1985; Ives, 1991; Ives, 1993; Hartley, 1990; Knott & Goodall, 1991). The author is indebted to British Gas (Scotland) for all of the information and gas mains data that went into the creation of database model for Gas 'R' Us ltd.

9.1.3.1 System Extent

Gas 'R' Us ltd has a computer system based on a distributed network with several interconnected processors and data stores at different locations. The central processor and data storage facilities are based at a central HQ, with satellite processors with local storage facilities sited at each district office (see Figure 9-1). The distributed configuration means that two copies of the regional mapping database are stored - one at the central site, the other at the district, with each district holding a subset of data relevant to its geographic area. The distribution of data allows the district to operate without dependence on the central system other than for system management activities such as backup and transfer of updates from districts to the central database. It is proposed that the corporate database created for phase II will represent the database system at central HQ level.

Within the system there are two distinct networks. The wide area network (WAN) which connects the district sites to the central site, and the local area network (LAN) connecting all local processors, storage devices and peripherals such as printers and plotters within a given site.

The central processing environment is needed for large volume, data processing and system management tasks and is an ideal location for a parallel GIS environment. There is need for a very fast turn around of large volumes of data from both central and district applications from the updating and maintenance of spatial and gas main data. The data once available in the system will be used by many different GIS and MIS applications. Fast, secure, multi-user access is needed to support the data requirements of all of the district sites. Data will also be received from personnel such as repair crews, surveyors, and meter readers, to name a few, able to record information in hand held systems to be sent back to base for processing. The parallel database may also have to support many transaction processing functions, particularly involving the customer billing system. For large customers, gas bills may calculated on a fifteen minute basis, rather than quarterly. In the future it is envisaged that bills will calculated on a minute-by minute, or even second-by-second basis, creating millions of transactions a second. Consequently, both the changing price of gas, and the customers' gas usage must be monitored and adjusted accordingly. In an integrated IT system all of this information will be accessible to the GIS through the parallel database.

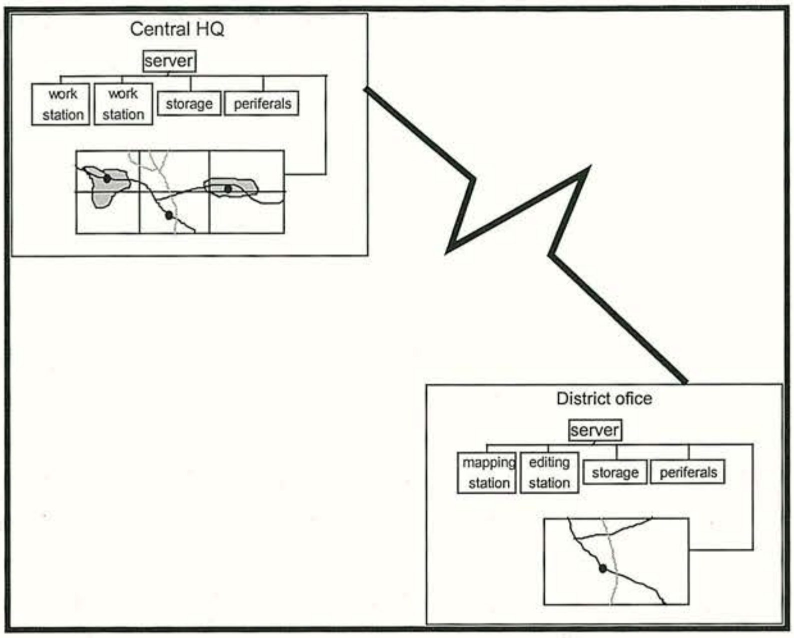


Figure 9-1: Central HQ and District Structure

The system software consists of the following components: the operating system; the GIS applications; and database software.

9.1.3.2 GIS Role

The GIS software provides the geographic database with comprehensive facilities to enable the user to manipulate the gas main network, to allow:

- **Geography take-on**, to load and update the base digital maps.
- **Mains and Plant take-on**, to digitise mains networks and associated plant and create links to external databases.
- **Mains and Plant Maintenance**, to update the mains and plant take-on with full facilities for creation, amendment and deletion of facilities.
- **Plotting**, to provide hard copy reporting facilities i.e. plots, microfiche, project drawings

- **Applications**, for example, network analysis, inter utility exchange, leakage surveys (Hartley, 1990).

These functions all require the GIS to access external data from a variety of database sources.

9.1.3.3 Scope of GIS Environment

The system must provide facilities for: the engineering management of the gas mains network; the management information and control; the user task management; and the distributed data management.

It is important that engineers have fast, efficient, access to all relevant data for issues they are dealing with, e.g. upgrading gas mains networks, before decisions have to be made. Decisions may have to be made at short notice, and therefore, the system should provide the ability for *what if* questions across the whole expanse of data, from the full range of attribute data on a gas main, the job history on the gas main and similar mains, to network flow and pressure data. The company already has in existence a series of databases for corporate and engineering needs; there are also network analysis systems available. The requirement of this system is that it provides efficient linking to external systems to provide the necessary level of service for engineers.

The GIS role and environment described above are those actually found in large utility companies and are based on the experience of British Gas.

The integrated computing facilities must also provide all the information required by management. For example:

- the control of data take-on and update, to show the status of work in progress and work about to be started.
- the status of individual jobs, work orders, or the projects. Integration with existing mainframe control facilities should also be maintained.
- gas network information.

Finally, the system must cover the control of user tasks, providing the necessary data controls so that users can only enter or update data which satisfies the control criteria. For example, a low pressure main would not be connected to a medium pressure main; a valve only has a single active inlet and outlet main. Therefore, validation facilities to assist the user are required to assess the network, for example, facilities such as 'free end' network traces that identify where mains have not been connected into the network despite appearances. The emphasis is on these controls to establish and maintain data integrity (Ives, 1991).

9.1.3.4 System Organisation

The system is designed to allow multiple users access to the specific data required for the particular task in hand. Access to the data is in layers, grouped into layers or sub-layers so that different categories or groups of data can be manipulated independently. The success of the system hinges on fast and efficient access to the data.

User interfaces to the system are dependent upon the particular operations to be performed. They range from highly customised, graphics and menu driven interfaces, e.g. the index map displaying all the maps for that district, colour coded to indicate its status, to command driven interfaces for system administration. The system must be able to support the data requirements of each of the interfaces.

9.1.4 The Data Model

9.1.4.1 Gas Company Requirements

The database model is required to provide support for the data input and output applications described above. The tasks fall into two main areas: the inputs take the form of creating and updating data held in geographic databases; the output from the geographic database forms goes to a variety of applications, from high-level, graphical interface applications to data mining operations. The data are held in a number of databases, from proprietary GIS formats to general database applications.

Integration

The first is the integration of data from many different sources, both internal and external to the company (and district offices, council, other utilities, health boards, and police). For the system to work efficiently and the company to fulfil its aims and objectives there must be access to a variety of data sources.

Speed

The second is speed. The system must respond quickly to requests, particularly to complex, *ad-hoc* queries run in emergency situations where an almost instant response is required. In an environment where a variety of operations are all running at the same time, the data model must be designed to support fast access to data to reduce the chances of bottlenecks developing. This may mean that the traditional rules for relational data have to be relaxed.

Data Processing

Finally, there should be support for different types of data processing. Mixed in the datamodel are databases that support transaction processing, and those supporting more ad-hoc type queries. There should be no detriment to performance when, for example, both transaction processing and complex query execution occur at the same time.

9.1.4.2 Research Requirements

From the descriptions above a number of requirements for the datamodel can be extracted. These are:

- The datamodel should supply a database of sufficient size and complexity to provide a realistic environment for performance tests.
- A variety of data tables should be available, from simple look-up tables to much larger, complex tables.
- Data tables of different types should be available for different types of query processing, particularly transaction processing and complex query processing.
- The datamodel should allow access to data held in different locations, such as the ORACLE database, ARC/INFO and Smallworld.

9.1.5 Model Design

The model was designed using CASE*Designer™, an Oracle Corporation product which provided a comprehensive environment for modelling organisations and includes a graphical Entity-Relationship (E-R) environment. This model was selected because it represented the real world in a more natural manner than some of the other data models in existence, and it translates well into a relational database environment (Chen, 1976; Stonebraker, 1994). In the entity-relationship model about Gas 'R' Us ltd only relevant information concerning entities and relationships pertaining to the company were recorded. The following sections briefly describe E-R modelling and CASE*Designer™ terminology and graphics.

9.1.5.1 Entity Relationship Models

E-R models, based on set theory and relation theory, consist of entities and relationships which incorporate some of the important semantic information about the real world, and use a special diagrammatic technique to aid in database design (Chen, 1976). The entities and relationships recorded at this level in database design are conceptual objects in the mind.

Entities

An entity is a *thing* that can be distinctly identified and is of relevance to the organisation, such as for example, a customer, a street or a gas mains. Each entity has a series of attributes, which are features describing the entity. For example, each customer has a name, which can be broken up into title, initials, and surname. Each entity also requires a unique identifier. This consists of one or more attributes which when combined form a *primary key* for each customer.

Relationships

Relationships are connections or mappings between entities, for example, "`customer - mains`" is a relationship that exists between the customer who is a gas consumer and the mains which is the mechanism for delivering gas to the customer. Relationships are defined between entities using the entity primary keys. The primary keys of the two entities in the relationship combine to form the *relationship primary key.*

9.1.5.2 Entity-Relationship Diagrams

Entity-relationships are usually represented in diagrammatic form. There are a number of different methods for representing these. However, they all show the entities, the relationships, and the type of relationship i.e. one-to-one, one-to-many and many-to-many. In this thesis the method used in CASE*Designer™ will be adopted to describe the datamodel for Gas 'R' Us ltd. because the main body of design was undertaken with this application.

CASE*Designer™ E-R Diagram Method

This E-R diagram Figure 9-2 uses rectangles to represent the entities, a variety of line types

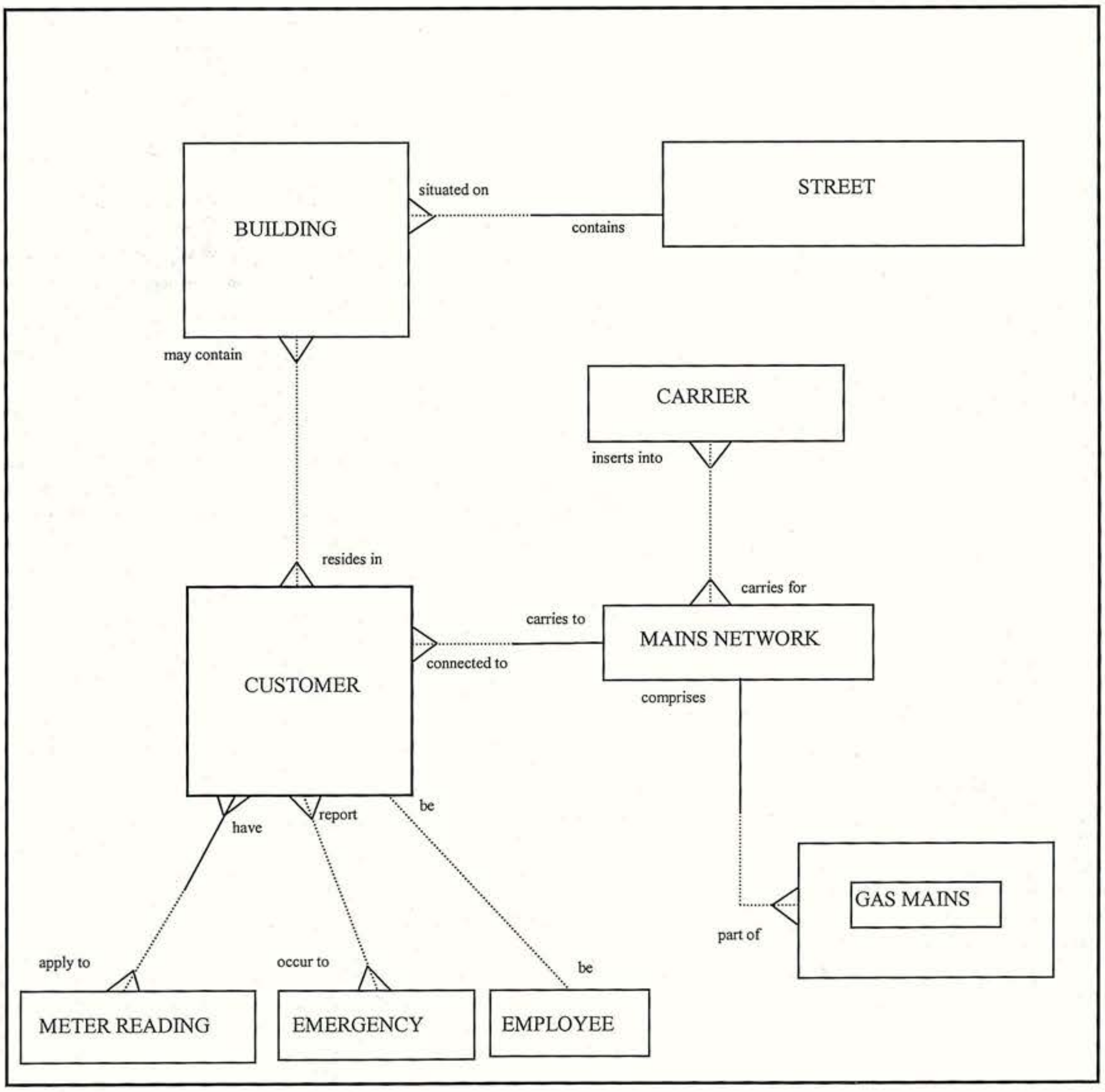


Figure 9-2: An Example of a CASE*Designer™ E-R Diagram

to represent relationship mappings, and textual descriptions to denote the relationships between entities.

The mappings between entries are used to convey a number of pieces of information:

- the nature of the mapping i.e. one-to-one, one-to-many, many-to-many. A crow's-foot symbol is used to indicate many mappings (Figure 9-3)

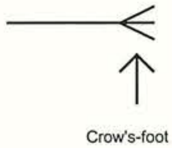


Figure 9-3: Crow's-foot symbol

- the status of the relationship i.e. mandatory or optional (Figure 9-4),

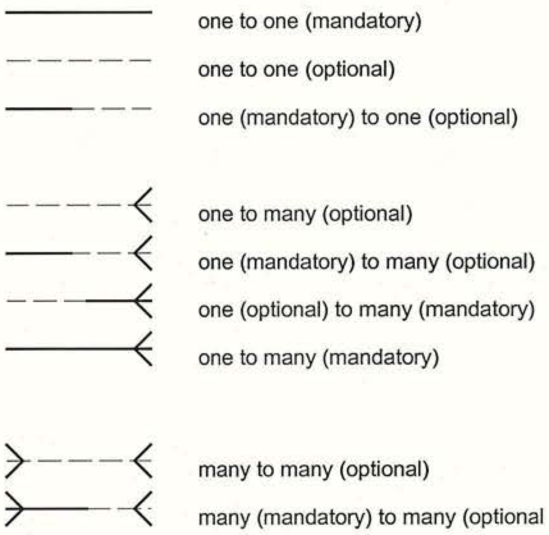

one to one (mandatory)
one to one (optional)
one (mandatory) to one (optional)
one to many (optional)
one (mandatory) to many (optional)
one (optional) to many (mandatory)
one to many (mandatory)
many to many (optional)
many (mandatory) to many (optional

Figure 9-4: Types of Relationship in CASE*Designer™

9.1.6 Data Sources

The data model allows for data to come from a number of different sources and different systems. These include GIS applications, remote databases and sources of information external to Gas 'R' Us ltd., e.g. other utilities and local government and data collecting agencies. The model is also flexible in the location of data. The parallel server is acting as data warehouse collating information from other locations of the company to a main data store and making it available for analysis and interpretation. It is feasible that the mains network will be held in a GIS on one system, buildings information on a different system, and customer information on yet a third system.

Bringing the data together in one place on a powerful parallel server that is accessible to all parts of the company brings many benefits. It makes the data a corporate resource while keeping responsibility and data ownership with individual departments, increases the sophistication and complexity of analysis possible, keeps data up to date and solves some of the data duplication problems.

9.2 The Data Model for Gas 'R' Us ltd.

The data model consists of 37 separate entities simulating many different areas of a utility business (Figure 9-5).

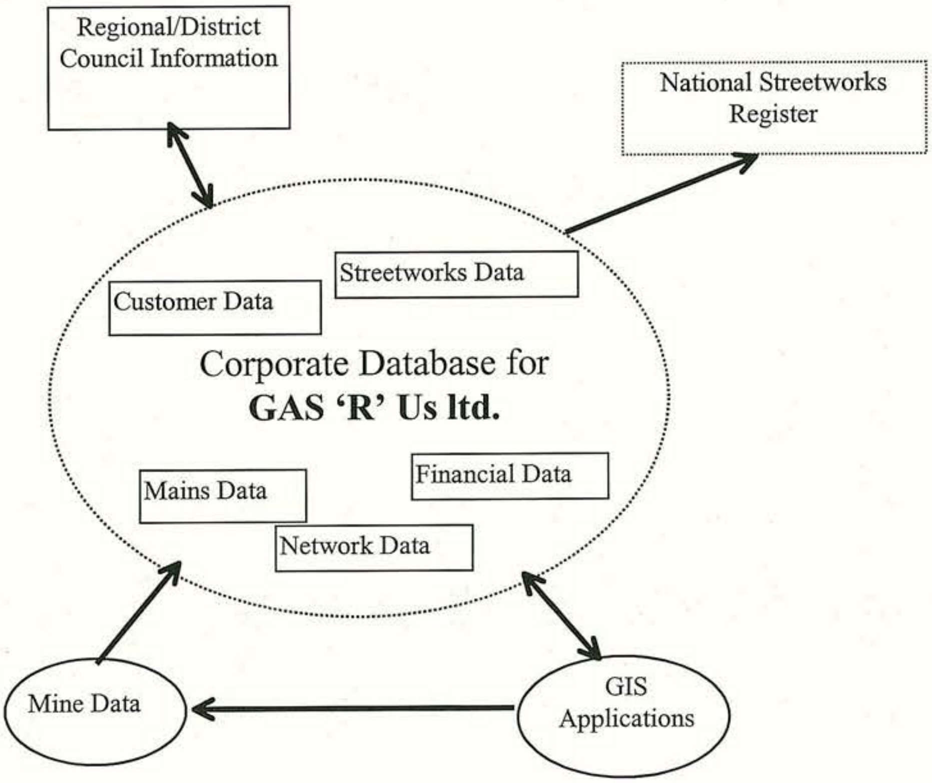

Regional/District Council Information
National Streetworks Register
Customer Data
Streetworks Data
Corporate Database for GAS 'R' Us ltd.
Mains Data
Financial Data
Network Data
Mine Data
GIS Applications

Figure 9-5: Database Model Overview

Each entity describes a particular, unique item of the business. For the purpose of the model, for example, the **customer** entity represents all the information about customers, the **building** entity represents all information required about the houses and factories gas is supplied to, **street** contains all relevant information about streets and so on. The entities can be grouped together to describe functions of the organisation, such as customer services, personnel, gas appliance services.

There are ten distinct groups of entity within the model. These are mains network, streets, street works, buildings, customers, employees, vehicles, appliances, external information and specialist contractors (Table 9-1). Although this is a rather simplistic view of a large corporate business, it incorporates enough complexity to simulate the types of requests demanded of a corporate information system. It also provides enough variety of data for both long, complex GIS queries, and the shorter, more rapid transaction processing queries.

Table 9-1: List of Entities that Comprise the Database Model

Entity	Associated Entities
Gas Mains Network	Gas Main, Control Entities: Governor Station, Injection Point, Siphon, Carrier
Buildings	Building Use, Household
Customers	Complaints, Special Needs, Account, Service Contracts, Installation, Meter, Meter Type, Emergency, Charges
Streets Works	Schedule, Completion, Plans, Originator/Recipient
Street	Town, County
Employees	Pay
Appliances	Appliance Type, Components, Appliance Maker
External Links	Census Data, Council, Credit Ratings
Vehicles	
Specialist Contractors	

The following subsections detail the main groups of entity found in the model. The complete model diagram is complex and difficult to read, therefore, each section contains an extract of the data model pertinent to that section as well as list of attributes for the main entities of the model.

9.2.1 Mains Network

The main network is the foundation on which Gas 'R' Us ltd is built and is their primary asset. An accurate representation of the network is essential to the company. Engineers planning new mains need to know the precise location of other gas mains in the area (as well as other utility mains or cables). Maintenance engineers require accurate information to locate relevant mains. Emergency engineers need to locate problems quickly and accurately to carry out repairs before an explosion occurs. Suppliers need to know the mains network structure to ensure that they are delivering their gas to the correct customers. The gas company requires accurate information to bill customers, monitor the performance of the network, make strategic decisions, and draw up maintenance schedules.

Other entities connected to the gas mains are control mechanisms such as valves, injection points, siphon points and governor stations. These all contribute to the maintenance of the network, for example, allowing engineers to shut down particular sections of a network, and monitor the flow of gas at various points. These are important features in the mains network which must be represented in the data model (see Figure 9-6).

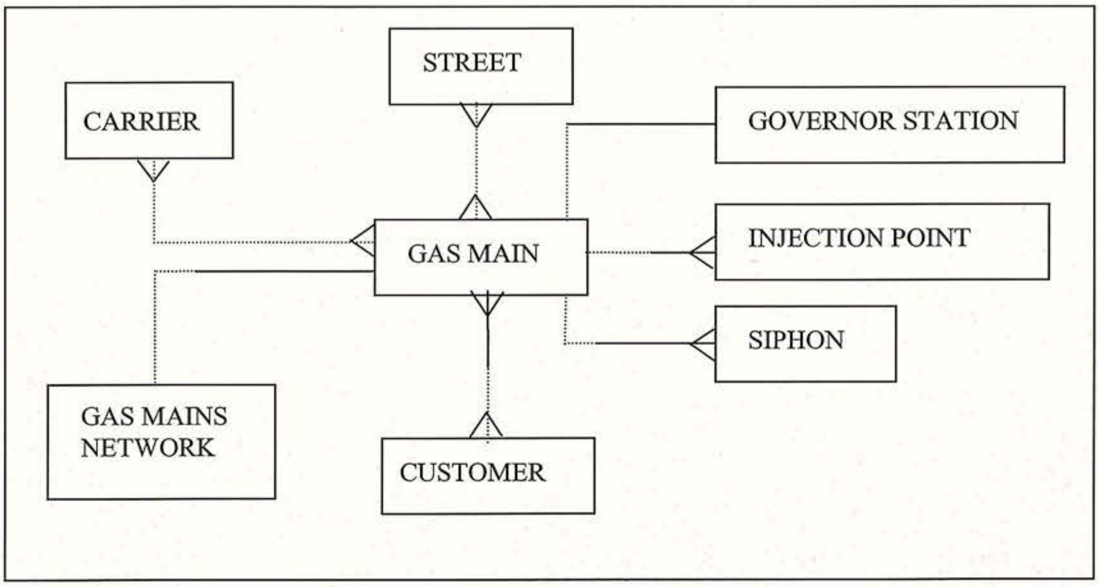


Figure 9-6: Entity Relationship model for the Gas Mains Entity

The central entity of the mains network are the individual gas mains. The attribute data for individual gas mains is stored in, what British Gas call, the Mine Database. The database contains full descriptions of individual gas mains. This includes physical attributes, such as length, material and diameter, street location, grid reference, and comments for engineers. Data from the Mine database were being made available to corporate GIS systems in British Gas through the Digital Records project. The section relating to the mains network for Cumbernauld was supplied with paper maps for use in this research project. An extract of the Mine database (Table 9-2) replicated in its original form, has been included to demonstrate the task of extracting information from it. A pile of paper approximately two inches thick was supplied from British Gas (Scotland)!

For the purposes of the mains network model the physical attributes of individual gas mains were stored with the spatial data (see Figure 9-7) and links were through to the Mine Database (see Table 9-2) using the *mains-id*, unique number assigned to each individual main.

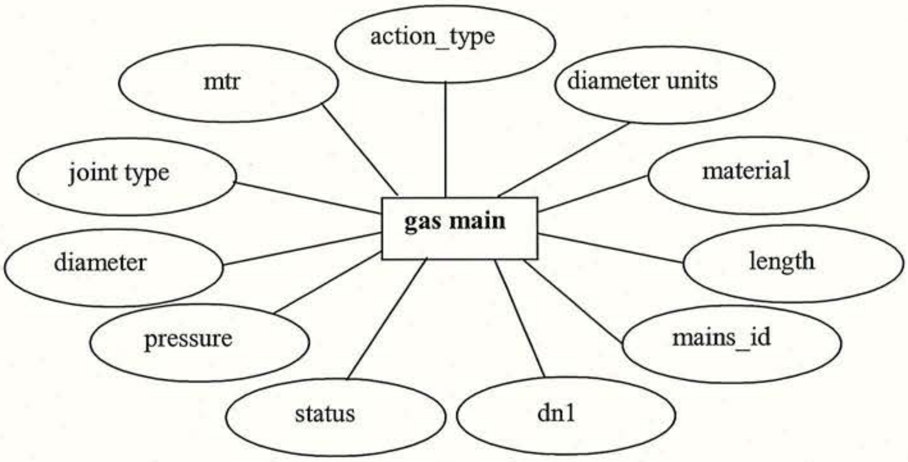


Figure 9-7: Gas Main Attributes

Table 9-2: Example of Mine Data

Grid street address	4044 mains & ST Ref OS Map	Main-ind mat sub-dist	diameter	length of main	limits odd limits even	mains-id
Derrywood Road Milton of Campsie	10 4 044 009260 009261 1 306734 NS6576NW	0 P 80	32	46	RDXSCOTT AVE INLT TO RRI	238907
Ferguson Terrace Milton of Campsie	10 4 044 009116 009117 2 306154 NS6576NW	0 P 80	25	17	SER TO 1	170601
Lochiel Drive Milton of Campsie	10 4 044 000022 000125 1 306499 NS6576NW	0 P 80	125	320	GOV INLET BALDORAN	169536
School Lane	10 4 044 009000 009001 1 306736 NS6576NW	0 P 80	63	87	ESL TO ESH	170124

The attributes for the gas main were derived directly from information from British Gas (Scotland) and are an exact replication of data, available from either mains diagrams on paper maps, or the Mine database.

9.2.2 Streets

The street entity contains all information the company requires regarding streets (see Figure 9-8). Information about streets is stored for many purposes including: the location of individual gas mains, customer information, sales and marketing, and emergency planning. Below are a sample of the details that could be included. The company might also want to hold details about physical features such as the material of the street, width, depth of surface, and the type of area.

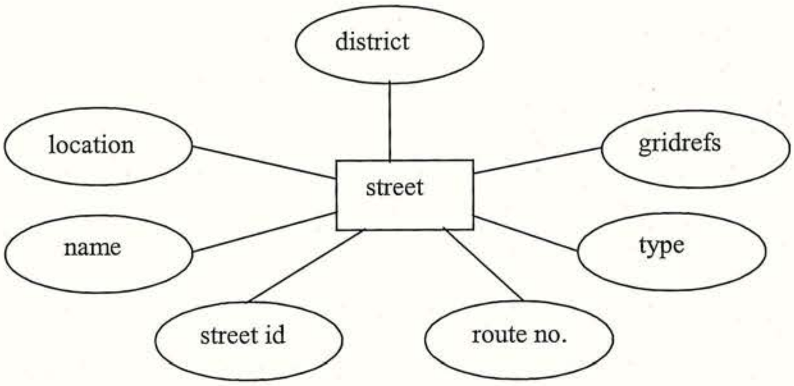


Figure 9-8: Attributes for Street Entity

Accurate street information is essential for utility companies for many reasons. A large proportion of the gas mains are buried below streets and pavements so they need to be located both quickly and correctly. This may prove rather problematic for information systems derived from paper maps as the grid references marked on the map may bear no relation to the reality on the ground.

The Street entity is connected to a number of other entities in the corporate datamodel (see Figure 9-9). These include information about the buildings lining the street, the town the street is part of. The entity is also connected to the gas mains buried under it and to the streetworks entity for future planning of gas mains.

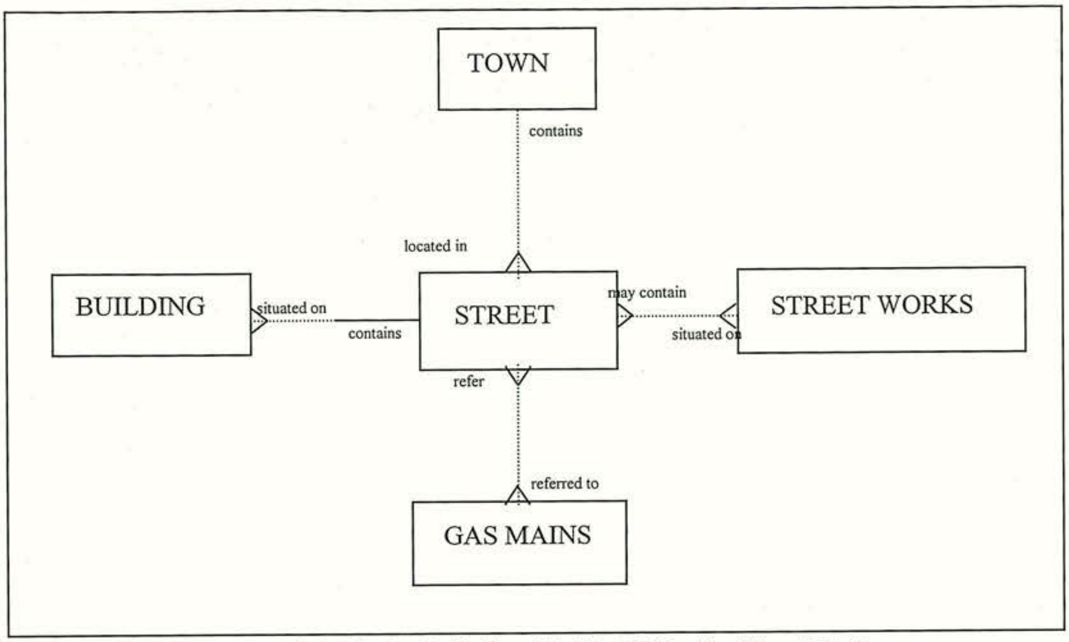


Figure 9-9: Entity-Relationship Model for the Street Entity

9.2.3 Street Works

The streetwork entity represent the part of the corporate datamodel dealing with all the communications and data associated with streetwork notices. It would be useful for a utility company to be able to process this information quickly to send out to other organisations. The individual attributes of the streetwork entity (see Figure 9-10) comprise all the information available about individual streetworks. The information required may be multimedia, including images and scanned documents as well as text.

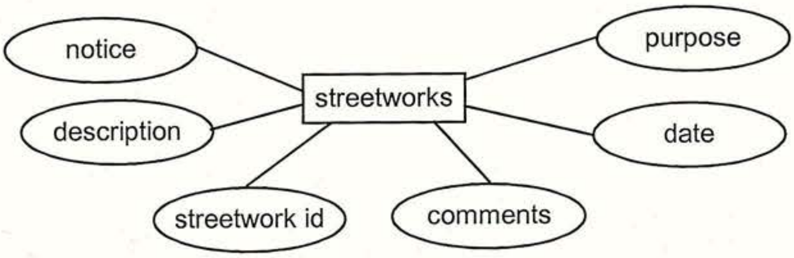


Figure 9-10: Attributes for Streetwork Entity

There have been a number of experiments to make this exchange of information through electronic communication and it is likely with the development of extranets that work in this direction will continue to progress. The streetwork model depicted in Figure 9-11 allows for the several stages of operation from the original drawing up of plans for streetworks to recording details of these who should receive the information. There are also links to spatial

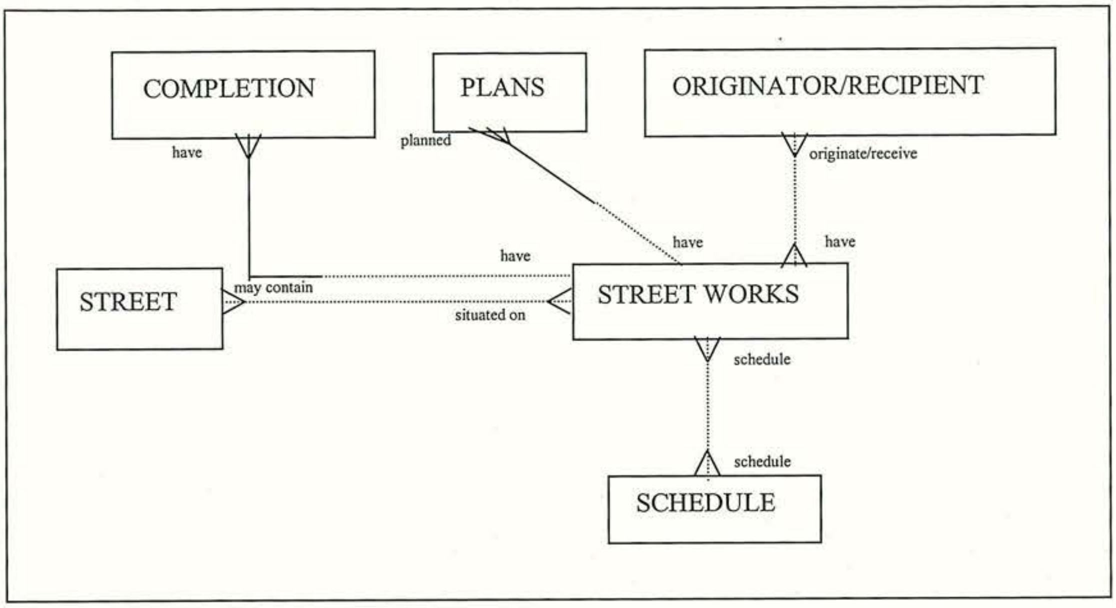


Figure 9-11: Entity-Relationship Model for the Street works Entity

and attribute data about the streets themselves and the surrounding area, which could also include links to the council for detailed road engineering data. There is also a link to a scheduler system that is used to schedule and allocate both work and staff to ensure that projects take place at the appropriate time with enough staff to do the job.

9.2.4 Buildings Entities

The building entities describe the installation sites where gas is consumed. Detailed information about each site is necessary especially in cases of emergency when gas engineers need to know what awaits them at the other end. It is possible to incorporate photographs, building plans, video clip into the model to provide a very detailed picture of each building, especially for the large commercial users of gas.

The buildings entity (see Figure 9-12) is a central object shared by any organisational function that requires an address. This includes sales, marketing, personnel, and payroll. The model was designed in this way so that address data need only be stored once, but are accessible to all. This means that everyone is working from the same updated version of the address list, which should reduce the number of instances where communications are sent to customers needlessly. This type of strategy should reduce customer complaints as well as save both time and money.

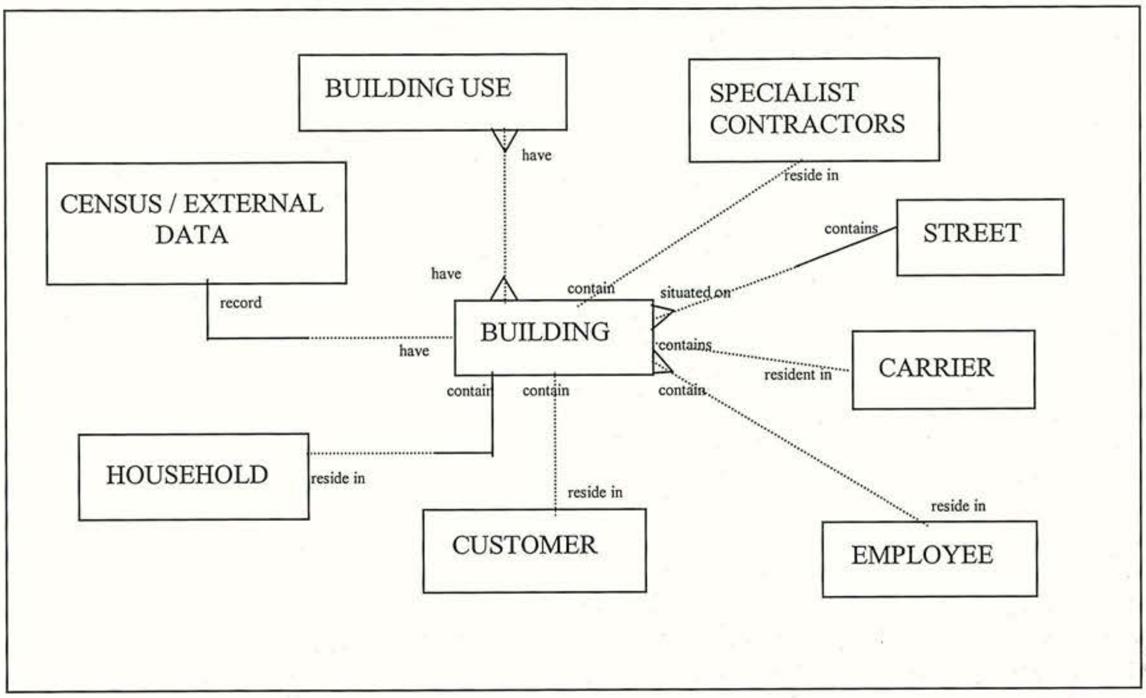


Figure 9-12: Entity-Relationship Diagram for Building Entity

9.2.5 Customer Entities

The customer is one of the main focal points of the organisation, customers (gas consumers) are the foundation of the business and generate a high proportion of work and profit for the organisation. The customer entity consists of only those attributes that directly identify and describe the individual (see Figure 9-13). This forms a small cluster of information that could also include sex and race if this were thought appropriate.

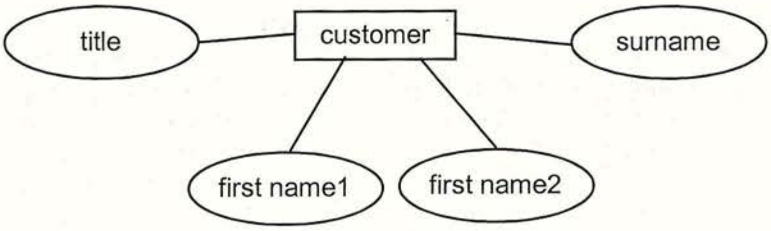


Figure 9-13: Attributes for Customer Entity

Along with the building entity, the customer entity is a central piece of information that links to many other entities in the datamodel (see Figure 9-14).

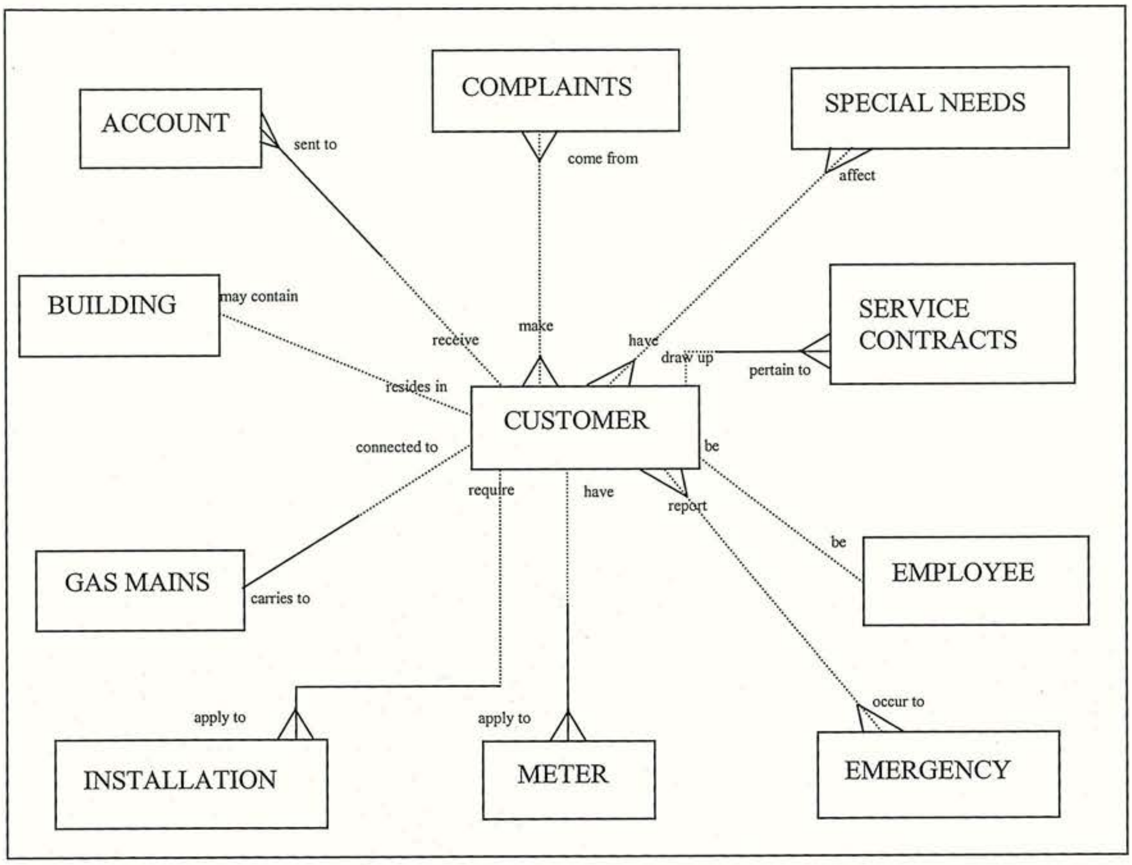


Figure 9-14: Entity-Relationship model for Customer Entity

Based on information contained within this entity group a transaction processing task can be created to process billing information.

Bills are processed and sent out quarterly to the customer based on the number of units of gas consumed and the price of a unit of gas. For the purposes of this research the price of gas is assumed to be constant over the quarterly period; however it is possible to include the fluctuating price of gas into the equation and bill the customers accordingly.

9.2.6 Employee Entities

The employee section of the entity model holds information about individual employees such as name, status, and salary (see Figure 9-15). There is a link to the customer tables because it is highly likely that many of the employees will be customers of Gas 'R' Us ltd.

and would be entitled to a discount. This section is partly designed for a modest transaction processing routine to calculated the pay of individual employees using a simple update function based on salary level and number of days worked. It would be possible to include factors such as number of hours worked per week, bonus payments, sick leave, special leave, and expenses occurred throughout the week or month, to extend the transaction processing function.

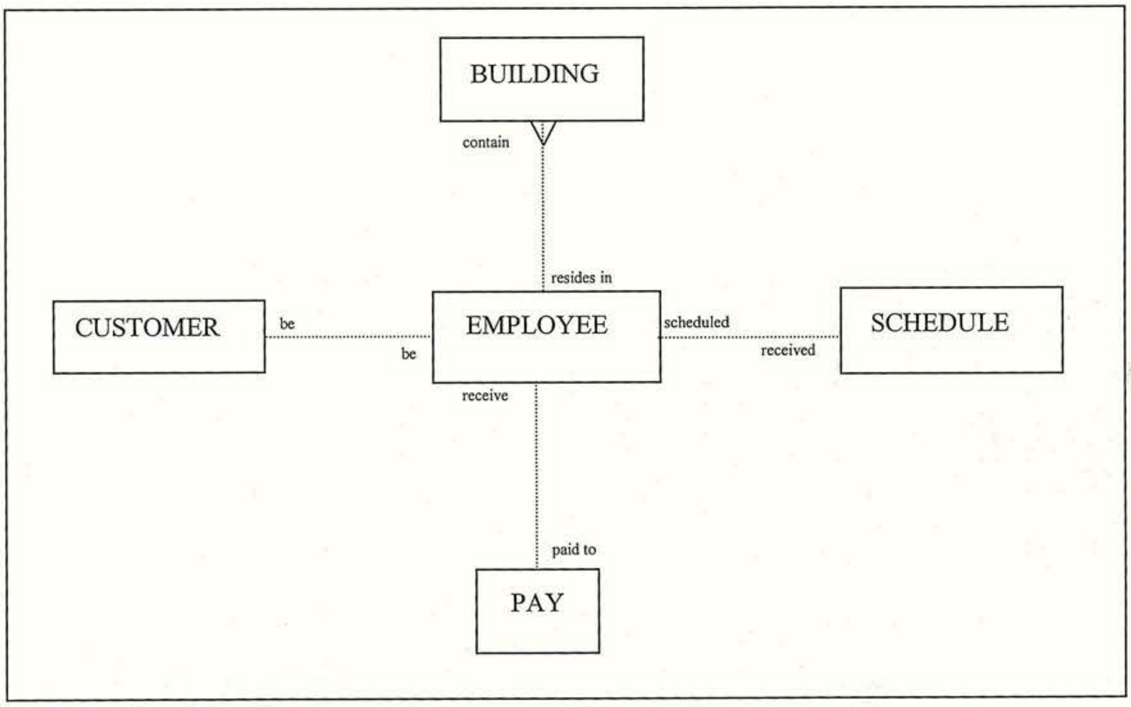


Figure 9-15: Entity-Relationship diagram of Employee Entity

9.2.7 Appliances

The appliance entities are there to represent another facet of the organisation, i.e. that of selling and servicing gas appliances for central heating systems, gas fires and water heaters. This part of the model provides facilities for developing a series of stock control functions, requiring the level of stock to be monitored and new stock ordered when necessary. The tables in Figure 9-16 also form part of a much more complex query that accesses data from many of the tables in the model (detailed in chapter twelve).

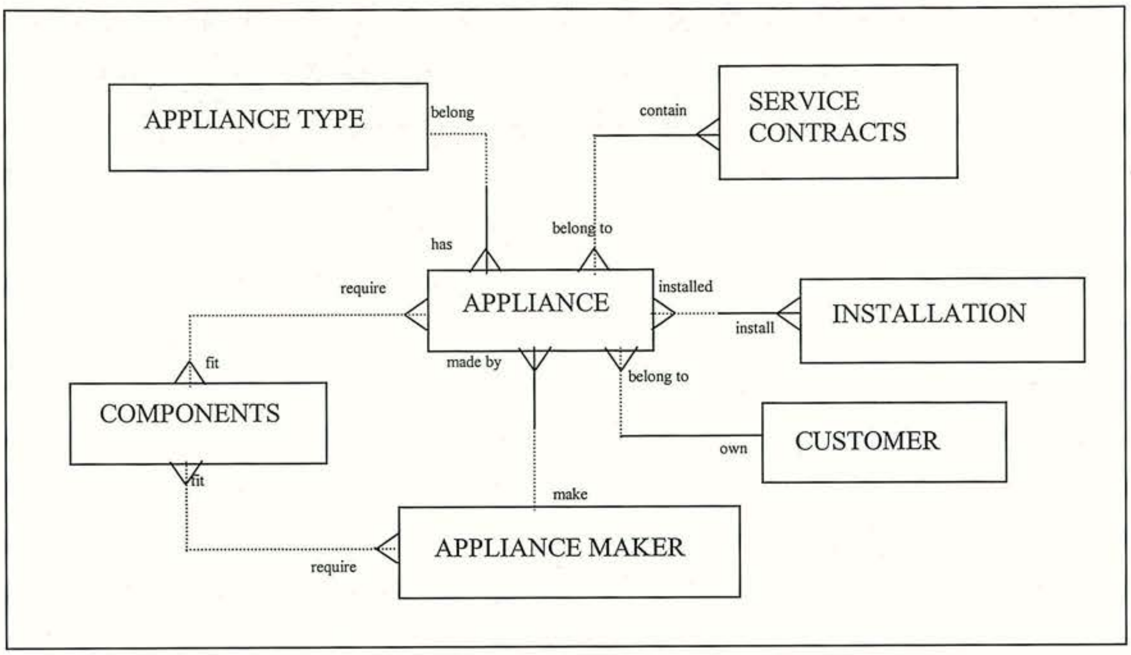


Figure 9-16: Entity-Relationship Diagram for the Appliance Entity

9.2.8 Summary

The entity-relationship model has been built around a company called Gas 'R' Us ltd. This fictitious company is based on British Gas, with many of the company requirements for an integrated corporate GIS influenced by their requirements and experiences of implementing the Digital Records project. The role of the data model is threefold. Firstly, it represents a selection of the main elements of an integrated corporate database. This includes spatial and attribute data for the gas mains network. Secondly, it provides an environment for creating both hybrid and integrated corporate GIS models where spatial data for the mains network can either be stored in native GIS formats or within the corporate database. Attribute data associated with the mains network is stored in the corporate database. Finally, it provides an environment where complex SQL queries with a large number of joins can be constructed and run, as well as those queries required for transaction processing. The entity-relationship model was designed to provide a database of a suitable size for performance testing using much more realistic data than that of the pilot database. The entities were created around the data they represented. The attributes of each entity were chosen because it was felt they were appropriate for that particular entity, and size was determined by the data. In contrast the pilot database was designed as a number of fixed length tables with the data created to fit the table requirements.

9.3 Database Creation

The database was created using code generated by the CASE*Designer™ software when the data model was complete (see Appendix B). Before the code was generated a series of integrity checks were made using the modelling software. The checks examined the validity of the relationships between entities, and looked for inconsistencies with entities and attributes. These are detailed further in Appendix C. The code once created was modified slightly to remove some of the constraints placed on the tables to facilitate the loading of data into the system. The data loading process proved to be rather by trial and error, particularly at the start and therefore the constraints were removed.

9.3.1 Data

The data in the database are all fictitious, created by a series of C programs written for the purpose (see Appendix D). The database was partially populated to allow a series of transaction processing queries and complex queries to be performed (Table 9-3).

Table 9-3: List of Table Names and Size

Table Name	No. Rows
Appliances	2 million
Buildings	1 million
Building Uses	10
Customers	1 million
Employees	5000
Meters	8 million
Pipes	3200
Schedules	2000
Streetworks	100
Streets	212
Towns	8

In total the database contained approximately 15 million rows of data, held in 10 different tables.

9.3.1.1 CUSTOMER/EMPLOYEE Table

The CUSTOMER table was populated by taking a digital download of the University of Edinburgh telephone directory and extracting all of the surnames. A program then randomly combined these surnames with a title - Mr, Mrs, Ms, Miss, Dr, Father or Prof., and a couple of letters to form the initials. The program was weighted so that only a small percentage of *Fathers* and a slightly larger percentage of *Drs*. and *Profs.* were generated. The rest were randomly split between the more common titles.

The program was used to generate one million customers for Gas 'R' Us ltd. Once the list was compiled a check was then made against the University Telephone Directory to ensure that the program had not inadvertently created a *real* person.

A similar process was used to create the EMPLOYEE table. However, it is possible that a high proportion of customers are also employees. Therefore, a large percentage of employees were taken from the existing customer database. The other employees were generated using the name generation program.

9.3.1.2 Building Table

The BUILDINGS table also contains fictitious data. The data were created using street names from the fictional town "Ankh-Morpork", created by Mr Terry Pratchett in the Discworld series. Postcodes were created using combinations of letters and numbers not currently used in the postcode directory.

BUILDING USES is a look-up table detailing a number of different uses for a building from domestic use through to heavy industrial use.

9.3.1.3 Meters

The METER readings table is the largest table generated and holds meter readings data over a two year period, and assumes that meter readings are made quarterly. The start number for the initial meter reading was picked randomly between a figure of 0 and 3000. The subsequent series of readings for each household over the period are incremental, randomly adding a figure between 75 to 2000. To make the data slightly more realistic the meter reading figures could have been dependent on the building use. However, there was little time to add such refinements to the generated data.

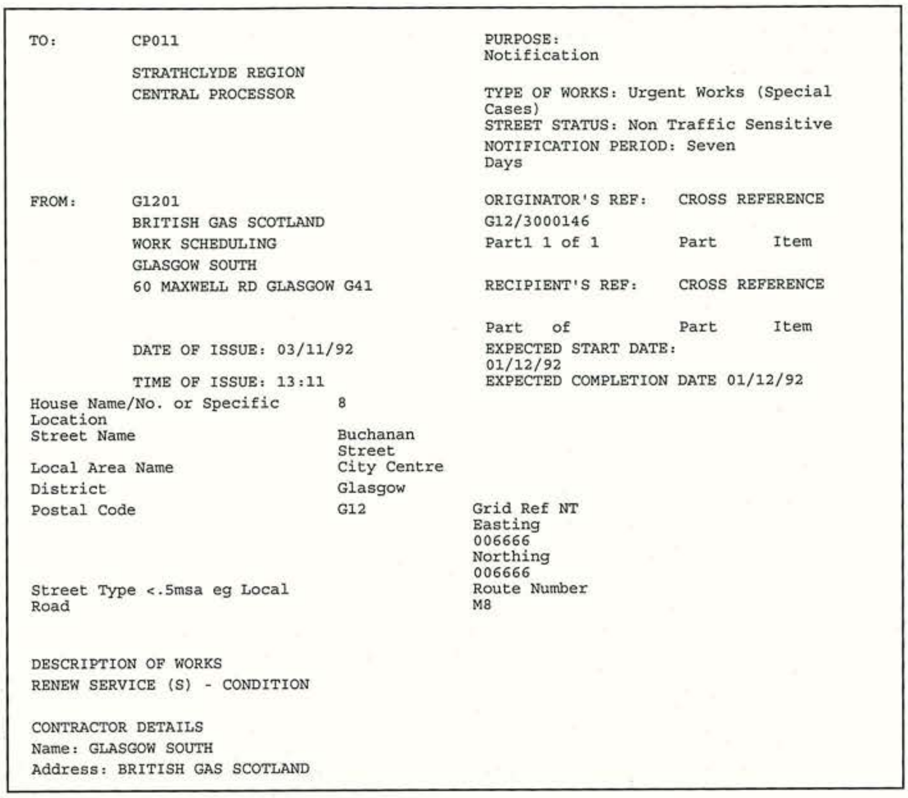
The ACCOUNT table was filled using the output from a transaction processing query (see example below). The query accessed meter reading details, calculated the bill for each customer for a particular quarter and stored all relevant data in the ACCOUNTS table.

```
select cust_id,a.mete_reading,b.mete_reading, b.mete_reading-
a.mete_reading
from meters a, meters b, customers
where ((cust_id = a.mete_id
       and a.mete_date between '01-JAN-93' and '31-MAR-93')
and (cust_id = b.mete_id
     and b.mete_date between '01-APR-93' and '30-JUN-93'))
```

Example of a transaction processing query

9.3.1.4 Streetworks

The data for the STREETWORKS table was derived from an example of a streetworks notification form from British Gas Scotland. Figure 9-17 shows a replica of a street work notification form. The form is distributed by the utility wishing to begin road works, to other utilities and the local government to notify them of their intentions. The recipients are then obliged to notify the utility issuing the notice of any known mains or cables in that area.



```
TO:       CP011                                PURPOSE:
                                               Notification
          STRATHCLYDE REGION
          CENTRAL PROCESSOR                    TYPE OF WORKS: Urgent Works (Special
                                               Cases)
                                               STREET STATUS: Non Traffic Sensitive
                                               NOTIFICATION PERIOD: Seven
                                               Days

FROM:     G1201                                ORIGINATOR'S REF:    CROSS REFERENCE
          BRITISH GAS SCOTLAND                 G12/3000146
          WORK SCHEDULING                      Part1 1 of 1         Part      Item
          GLASGOW SOUTH
          60 MAXWELL RD GLASGOW G41            RECIPIENT'S REF:     CROSS REFERENCE

                                               Part   of            Part      Item
          DATE OF ISSUE: 03/11/92              EXPECTED START DATE:
                                               01/12/92
          TIME OF ISSUE: 13:11                 EXPECTED COMPLETION DATE 01/12/92
House Name/No. or Specific     8
Location
Street Name                    Buchanan
                               Street
Local Area Name                City Centre
District                       Glasgow
Postal Code                    G12           Grid Ref NT
                                             Easting
                                             006666
                                             Northing
                                             006666
Street Type <.5msa eg Local                  Route Number
Road                                         M8


DESCRIPTION OF WORKS
RENEW SERVICE (S) - CONDITION

CONTRACTOR DETAILS
Name: GLASGOW SOUTH
Address: BRITISH GAS SCOTLAND
```

Figure 9-17: Example of a Streetwork Notification Form

9.4 The Gas Main Network

9.4.1 Main Network Elements

The main network for the company has been created from data kindly supplied by British Gas (Scotland). The data consisted of 113 1:1250 scale Ordnance Survey maps of the Cumbernauld area with individual gas mains drawn on by hand. The mains were supplied in both paper format (1:1250 sheets) and in a raster format compatible with the raster import routines in the Smallworld GIS. Each map file was named with its sheet reference and a ".lrd" extension e.g. ns7373sw.lrd. Paper copies of the sheets with and without mains identifiers were supplied. The maps date from around 1962.

The maps contain the following information:

- the location of each individual gas main
- the type of main - low, medium, intermediate pressure mains
- the material of the main - polyethylene, cast iron
- the date the main was laid
- the diameter
- the status of the main - used, abandoned, as-found
- the start and end node identifiers of the main.

Not all mains have all of this information attached, but most mains at least have the type, material and diameter. As well as mains the maps indicated the positions of valves, siphons, pressure points and other items sited on an individual main.

Each main has its own main identifier which is a unique identifier for each main in the network. The main identifier has been added by British Gas at a later date to the gas mains drawn on the maps as they require the information for their own digitised main network database.

Landline digital data (1992) from Ordnance Survey was also provided as a reference data set. The reference data set was essential due to the nature of the paper maps supplied by British Gas (Scotland). The maps suffered from a number of major flaws owing to their age and the number of times they had been copied. The problems were exacerbated by the area chosen. Cumbernauld was a new town and a large part of it was built after the maps were produced, making the accuracy of suburban features on the map rather suspect. Use of the Landline data will be explained in detail later.

9.4.1.1 Methodology

To create a mains network that could be used by a number of different GIS applications the mains had to be digitised as vectors from the raster maps provided by British Gas (Scotland). Smallworld was chosen for this conversion process for two reasons. Firstly, both the rasterised paper maps and Landline vector data could be imported into the system and manipulated. Secondly, Smallworld provided a sophisticated suite of on-screen digitising facilities.

The stages for creating the main network are as follows:

- create objects for raster data, Ordnance Survey Landline Data and main network in Smallworld
- create rules for objects
- test database - much more difficult to change at a later date
- import raster data and register each sheet to the National Grid Co-ordinates
- import Landline Data
- check accuracy of raster maps
- edge match of each sheet
- feature match between raster and Landline maps
- digitise mains and other objects with reference to both raster and vector maps
- export to other GIS applications.

9.4.2 Smallworld Model

Smallworld is an object-oriented GIS and requires the user to create a series of objects in which to hold the map data. For the Gas Mains model nine objects were identified from the paper maps. These were: main; valve; siphon; pressure point; carbo seal; governor house; governor; meter; and surface box. Smallworld also allows the user to develop a series of rules about how each of the objects interact with each other. The rules for the Gas Mains model were fairly simple. All of the point objects such as valves and siphons should be sited on a gas main. It is possible to have more than one point object sited at a particular point. Valves tend to be sited at the ends of mains, even though they are not drawn so on the maps (for sake of clarity).

Mains are defined by their mains identifier and their start and end nodes.

9.4.2.1 Mains

The gas main object has a number of attributes that describe each individual gas main, such as status, pressure and diameter. These are described below.

Status

Three types of status of main were identified. These are:

- **working mains** i.e. mains as shown on the map
- **abandoned mains** - identifiable by a scribbled line through them
- **as found mains**. These are mains that were discovered by one of the utilities when doing other work. The sections of main that have been located have been marked on the map, and identified as *as found.*

Pressure

Mains are designed to carry different gas pressures - the categories are:

- **supergrid** (sg) - to carry gas at very high pressure (over 7 bar) across country
- **intermediate pressure** (ip) - carries above 2 bar but not exceeding 7 bar
- **medium pressure** (mp) - carries above 75 mbar but not exceeding 2 bar
- **low pressure** (n) - does not exceed 75 mbar

A convention on the paper maps is not to show gas services to individual houses unless these are carried by medium pressure mains. Therefore, the main network for the imaginary gas company will not show delivery to individual houses. It is assumed that gas is delivered to each house that the mains network passes. Gas mains **are** attached to particular buildings in the Oracle database, even though those connections are not shown on the mains maps.

Material

Mains are made from different materials depending on the pressure of gas they are designed to transport:

- **steel** (st)
- **ductile iron** (di)
- **cast iron** (ci)
- **polyethylene** (pe)

Diameter

There are two units of measure used for the diameter of the main: inches and millimetres. For storage in the database it was decided to convert all diameters to millimetres, using the conversion of 25mm = 1 inch. Although this is not quite exact - it simplified the conversion process and moreover seems to be the method favoured by British Gas (Scotland).

Start and End Nodes

Each main has a labelled start node and end, the majority of which were marked on the paper maps by British Gas (Scotland). As this work was in progress at the time of digitising, not all nodes had been assigned start and end node identifiers. Therefore this information was only included where available. This information when complete could be used to verify the connection of individual gas mains to each other and thus provide a means of checking the integrity of the network.

Mains Identifier

Each main was assigned a unique mains identifier. Again this work was in progress during the project and although most mains had been assigned an identifier there were a few where this information was unavailable.

9.4.2.2 Valves

Valves have three identified attributes - state, size and position.

- **state** - valves can have two states: either open or closed. On the maps supplied by British Gas (Scotland) two types of valves appear, those that are normally left open, and those that are normally left closed.
- **size** - the size of the valve is occasionally identified if it is felt significant.
- **position** - valves tend to have two positions - they are either located on the surface or they are buried.

9.4.2.3 The Other Point Objects

All of the other point objects - siphon, pressure point, carbo seal, governor house, governor, meter, and surface boxes, have no visible attributes apart from location, which is stored as part of the object.

9.4.3 Creation of the Main Network

To allow the mains network to be ported to other GIS applications the Smallworld database has been kept as simple as possible. No behaviour has been defined for the objects except for display purposes.

A test database for the maps was initially created using only main and valve objects. This was done to test the loading, transforming and digitising of raster maps. Once this was seen to be working a full implementation of the database was undertaken.

For a GIS application in Smallworld (v1.9) it is necessary to provide MAGIK code to define the structures and special behaviour required by an applications Real World Objects (RWOs). It is conventional to provide an RWO definition file to define the structures and an EXEMPLARS file to define the behaviour. Two pieces of code were written in MAGIK to create the database called GAS_RWO.MAGIK, and GAS_EXEMPLAR.MAGIK. The code for these can be found in Appendix E.

The initial setting up of the database was fairly straightforward, following the instructions in the Customisation Overview Documentation provided by Smallworld (Smallworld, 1991c). However, once the database was set up it was not immediately clear how to go about making minor modifications to the structure.

Structural Changes

In simple terms, RWO code defines the structure of tables held in the database e.g. the main table. Exemplar code defines the interface to the database describing which attributes are visible and whether they can be updated. Therefore, changing an exemplar is easier than changing an RWO as exemplars can be removed and redefined without any effect on the structure of the database.

To change RWO definitions means changing table structure. To do this requires the table to be dropped and recreated - similar to tables in relational databases. When a table is dropped the exemplar is removed as well and will have to be reloaded when the table is recreated. This type of change should be done during the test phase while there are little data in the system or face losing a lot of work at a later stage. It is important to commit the changes and, when satisfied with the result, also save the image.

Data Entry of Raster Maps

The first step was to load in all the raster maps and align them with the national grid. It was necessary to align all the map sheets to check the accuracy of the alignment (to try to reduce errors due to the paper warping and distortions from the rasterising process) and identify those maps that required to be scanned again. Many of the mains cross map sheet boundaries - it is easier to digitise over the complete area rather than sheet by sheet.

The raster files took up approximately 80mb of disk space.

Methodology

Each raster map was taken individually, loaded into Smallworld using the load option on the raster menu, and then transformed. Transformation was done using all four corners of the map sheet and then checked for error – in this way errors were kept to a minimum.

There were a number of maps with no clearly defined corner points. These could not be added immediately to the database and had to be re-scanned.

Problems with Raster Data Entry

Somehow one of the boundaries of the raster maps became unset. This had a strange effect on the display of raster maps. When this particular map was selected the system would not display the maps surrounding the faulty one. It appears that an entry was placed in the index but not in the actual table. The solution was to write a procedure in MAGIK to identify records in the raster table whose value is unset. Those records were then removed and the raster sheet reloaded.

Symbols

It was decided to display the data in Smallworld as close to the original maps as possible. Therefore a number of symbols had to be created. The symbols were difficult to create as the procedures for creating them were not intuitive. Symbols are created using the style option on the "Themes and Styles" menu.

The following symbols were used for display purposes in Smallworld:

- Mains Colour
    - normal = blue
    - abandoned = green
    - as_found = magenta
- Mains Line Type
    - normal pressure = solid line
    - medium pressure = chain dash
    - intermediate pressure = long dash

    - supergrid = short dash
    - error = red/short dash
- Mains Symbol
    - 1 = normal/normal pressure
    - 2 = normal/medium pressure
    - 3 = normal/intermediate pressure
    - 4 = normal/supergrid
    - 5 = abandoned/normal pressure
    - 6 = abandoned/medium pressure
    - 7 = as_found/normal
    - 8 = as_found/medium
    - 9 = error

In order to show mains in different colours and line styles it was necessary to write a piece of MAGIK code to look at the main status and the main type, decide which symbol to choose, and return the integer code for the relevant symbol. The 2 attributes have a defined set of entries:

- Main Status
    - n = normal
    - ab = abandoned
    - as = as_found
- Main Type
    - n = normal
    - mp = medium pressure
    - sg = supergrid

- ip = intermediate pressure

The two files were called MAIN.MAGIK and VALVE.MAGIK and can be found in Appendix E

Loading in vector data

The loading of vector data, such as Ordnance Survey OS) OSTF (Ordnance Survey Transfer Format) maps, as background data followed a similar process to that of the raster maps. Smallworld required the user to define the specific features, and their numeric identifiers, contained in the maps, for example, different road types, rivers etc., and these were defined in an OSTF document published by OS (Ordnance Survey, 1988). The OSTF identifiers had to be linked to some Smallworld index numbers defined in the procedure "read_ostf". A number of difficulties were encountered due to bugs in the software but these were isolated and repaired. The OSTF files were large and took several hours each to load.

Digitising

Digitising the main network took eight weeks with two people working at it. There were a number of unexpected problems. It is possible for mains to span a number of map sheets. When mains were drawn in by hand little effort was made to make sure that the main ends matched up from one sheet to the next. When digitising a decision was made to make a best fit line rather than digitise to the end of one sheet then jump to the start of the next.

As referred to earlier, the maps were printed around 1962 when Cumbernauld was in its infancy. Many of the housing estates and retail parks were at the planning stage only. When they were built it was often not in the location indicated on the printed map. When using the paper maps this did not present a great problem as the map features are only visual clues to the user and all positions are relative. However, when transferring this data into a GIS each main is given a physical grid location. Using the OS vector data as a more accurate guide, the accuracy of the raster maps were examined. In some areas it was found that map features on the raster and vector maps were up to 40 metres apart. In such cases mains were digitised relative to the OS vector maps rather than the raster maps.

Transfer of Smallworld data to the Parallel Computer

A test database consisting of all the data structures and a very small amount of data was created. Two files . rwo.MAGIK and exemplar.MAGIK were created containing all relevant.

code. This was loaded up into a test version of Smallworld and a closed image created. These were then transferred to the parallel computer using FTP and dumped into an identical structure to that on the original machine.

The transfer process of Smallworld from the single node machine to the parallel machine was not a smooth process. This was partly due to the changing location of many of the files required by Smallworld, and also the change to a new version of Smallworld. Some internal tinkering with Smallworld was necessary to complete the transfer successfully. Once this was achieved a migration of spatial data from Smallworld to Arc/Info was achieved.

9.5 Summary

The test databases, a spatial database and an integrated corporate database created for phase II were required to provide facilities for substantial testing of integrated and hybrid corporate GIS models accessing data stored in a parallel environment. There were several stages to the development of the databases, including decisions about the corporate environment envisaged for the databases, the creation of a suitable datamodel and the construction and population of both databases. Decisions about the corporate environment were based on the British Gas digital records project, which was an extensive programme to convert all map data held by British Gas into digital form for widespread use throughout the company. It was important to reflect what roles and uses were envisaged for the spatial and attribute data to build a framework for the database model.

The design of the conceptual database model occurred in several stages, from the initial conception of entity groups such as customers, buildings and gas mains, and how they would fit together and interact, through to the creation of individual entities and their attributes. Some time was spent on ensuring that the database model was coherent and much use was made of the validation facilities in CASE*Designer^{TM}. Once the datamodel was complete it was converted into SQL code and loaded into Oracle. The database model represented all entities in the same location with links between different entity groups. However, in reality data was stored in two locations: the spatial data was held in Arc/Info and the corporate database in Oracle. External links were depicted to other data sources and it was expected these would be made through e-mail, or internet connections.

The final stage was the construction of data to fill the database. At this stage the conceptual model was split up into its individual components and developed into two separate databases. The spatial data were digitised using Smallworld and migrated to both Arc/Info and Oracle. The digitising process alone was a full time job for two people for two months. This did not include the learning curve for developing the Smallworld environment. Data for the corporate database were created using a mixture of real and computer generated data to simulate the mix of data expected in the database. The data had to be generated due to the confidentiality and data protection difficulties of using actual data from British Gas systems.

The database construction process was very complex and lengthy. There were many different software applications and environments to co-ordinate and manage and, as with many experimental projects, there were difficulties with the data, the software and the hardware which extended the length of development time considerably. The amount of data generated to populate individual database tables challenged the abilities of editing facilities and the storage media and it was necessary to make many adjustments to the parallel environment to accommodate the full database successfully.

The successful outcome of this stage of the project was a working corporate database, a GIS system with a substantial gas mains network linked to the database through a number of corresponding attribute tables, and an environment ready for performance testing.

The next stage of the research was to design a performance measuring environment. Performance testing in a parallel environment is complex because there are so many actions happening simultaneously. Due to the lack of parallel tools available for performance monitoring it was necessary to devise methods of collecting data from multiple nodes, the Oracle database and the GIS simultaneously. The performance tests envisaged would require all of these elements to work together in a parallel environment and therefore, two performance test harnesses were created. The first was for use with the pilot database, and the second for use with GIS/database interfaces. The next chapter describes the measurement environment and the performance test harnesses created.

10. Performance Measurement

10.1 Introduction

Sawyer (1993) described the second major hurdle of benchmarking as defining the measurement environment (the first was to define the workload!). Designing a suitable performance measuring environment for the project in a parallel environment was much more complex than in a sequential environment because there were many more things happening simultaneously, and monitoring tools for parallel processing were in their infancy. The majority of the facilities provided to monitor system and application performance were designed for sequential processing and were only capable of monitoring a single node at a time. Therefore, it was necessary to devise methods of recording the performance of the whole system as well as the individual nodes.

A further complication existed because the parallel environment had three separate levels of processing that should be considered, the SPARC nodes, the transputer nodes and the parallel disk farm. While the facilities for monitoring the performance of the SPARC nodes were relatively sophisticated, there were no direct methods available for measuring the performance of the transputer nodes or the disk farm, and their performance had to be inferred from the overall performance.

From the performance test specification there appeared to be two main areas where performance measuring was required. The first was in collecting performance statistics for queries run purely on the Oracle RDBMS. These were for testing response times for data loading, indexing, the effect of the buffer cache on query response time, and the effect of transaction mixes on system response time. The tests required that users log onto the database, perform the selected test and then exit. The second area was that of the GIS/database interface. This required statistics to be collected from the GIS as well as the database, and also that tests be started from within the GIS itself to measure the response times for GIS users.

The performance measuring requirements for the two different types of test were quite separate; therefore, two test harnesses were designed, one to be used with all tests devised to work with the database alone, and another to be used with Arc/Info to collect figures when using Oracle as an external database.

10.1.1 Database Performance Measures

The black art of database performance tuning has long been a contentious subject because there are so many areas where inefficiencies could occur. These range from the designers and users of the system: (Corrigan & Gurry, 1993) to the physical components, for example, memory, disks, and networks.

When deciding what to measure it is important to consider what the common problems usually are. There are four basic components of the machine environment which interact and affect system performance.

- Memory

  Memory bottlenecks occur when there is not enough memory to accommodate the needs of the application or group of applications. When this occurs excessive paging (moving portions of processes to disk) and swapping (transferring whole processes from memory to disk) start to happen.

- Disk I/O

  Disk bottlenecks occur when one or more disks exceed their recommended I/O rate. The disk drivers and controllers have a maximum I/O throughput, both in terms of I/O per second and blocks per second.

- CPU

  CPU bottlenecks occur when either the operating system or the software applications are making too many demands on the CPU. This is often caused by excessive paging and swapping.

- Network

  Network bottlenecks occur when the amount of traffic on the network is too great or when network collisions occur.

There were two areas of the system that had to be considered when undertaking performance monitoring of a parallel database, the system environment and the database itself.

Monitoring tools for the MCS were part of the operating system, SunOS. The database monitoring tools were part of the database management environment.

10.1.1.1 SunOS Monitoring Tools

The following tools with SunOS are available for monitoring the performance of the parallel environment: ps, vmstat.

These were standard tools for collecting statistics on CPU usage, interrupts, swapping, paging and context switching for each individual node on HYDRA. However, they did not allow the user to monitor all of the nodes simultaneously through a single interface. It was necessary to log on to each node individually and start the monitoring tool for that particular node. This was acceptable for a very small number of nodes, but monitoring even ten nodes became a very complex and difficult operation, particularly for interactive monitoring where it was necessary to have a window open for each node.

10.1.1.2 Monitoring Tools in Oracle

There were a number of monitoring and diagnostic tools available with the Oracle RDBMS to help the user examine the system and database statistics to tune the database to improve performance. The list below includes all the facilities that were used during the course of the project.

- MONITOR

A SQL*DBA facility for examining various system activity and performance tables

- SQL_TRACE

A utility that writes a trace file containing performance statistics

- TKPROF

A utility for translating the SQL_TRACE file into a readable form. It is also able to show the execution plan for SQL statements

- EXPLAIN PLAN

A statement that analyses and displays the execution plan for SQL statements

- BSTAT (begin) & ESTAT (end)

Scripts which produce snapshots of how the database is performing

- V$SYSSTATS

A view that contains a variety of processing and performance statistics which can be queried in a variety of ways.

The limitation with all of these tools was that while they allowed the user to monitor the performance of a wide variety of statistics pertaining to Oracle, they were specific to Oracle and only gave an indication of how the whole system was working from the viewpoint of the database. They did not, for example, help to monitor what sort of effect the disk controller buffer had on buffer caching in Oracle.

10.1.1.3 GIS Applications

Monitoring of performance in Arc/Info was done using the AML command &WATCH, which collected statistics and wrote them to a named file.

10.1.1.4 Elapsed time

The elapsed time for individual operations is a very useful measure of how the system is performing because unlike other more complex performance metrics it does not vary unpredictably as is possible when using monitoring tools from different software applications to collect statistics. It also the least complex metric to measure once it has been decided when it is appropriate to start and end the period of measurement.

Using elapsed time as an indicator of how the database is performing under a number of different conditions also provides a means of comparing results across all the performance tests to identify where bottlenecks are occurring. Once they have been identified it is then appropriate to start examining the performance in more detail.

10.2 Database Performance Testing Harness

The requirement for this harness was to collect various performance statistics for any given SQL query, or group of queries, run on any or all of the nodes on the MCS. There were a number of tasks that had to be completed to set up the environment for the performance test, including starting up an Oracle instance on the desired nodes and initialising start values for statistical variables before actually performing the test and gathering up the results (see Figure 10-1). The test harness was written in C, and used Pro*C commands to issue instructions to Oracle. The following sections detail the separate parts that make up the performance test harness.

10.2.1 Resource Usage Statistics

The resource usage module is used to collect statistics about the UNIX system using standard C programming language libraries. The module collects user and system time using the `getrusage` function which returns information about resource usage. A comprehensive list of resource indicators were collected, including:

- system time,
- user time - user and system time were converted to number of seconds used
- memory in use during program run
- number of page faults
- number of swaps
- block input and output
- messages sent and received
- number of signals delivered
- number of context switches

The results from the function were written out to file at the end of the performance run for analysis.

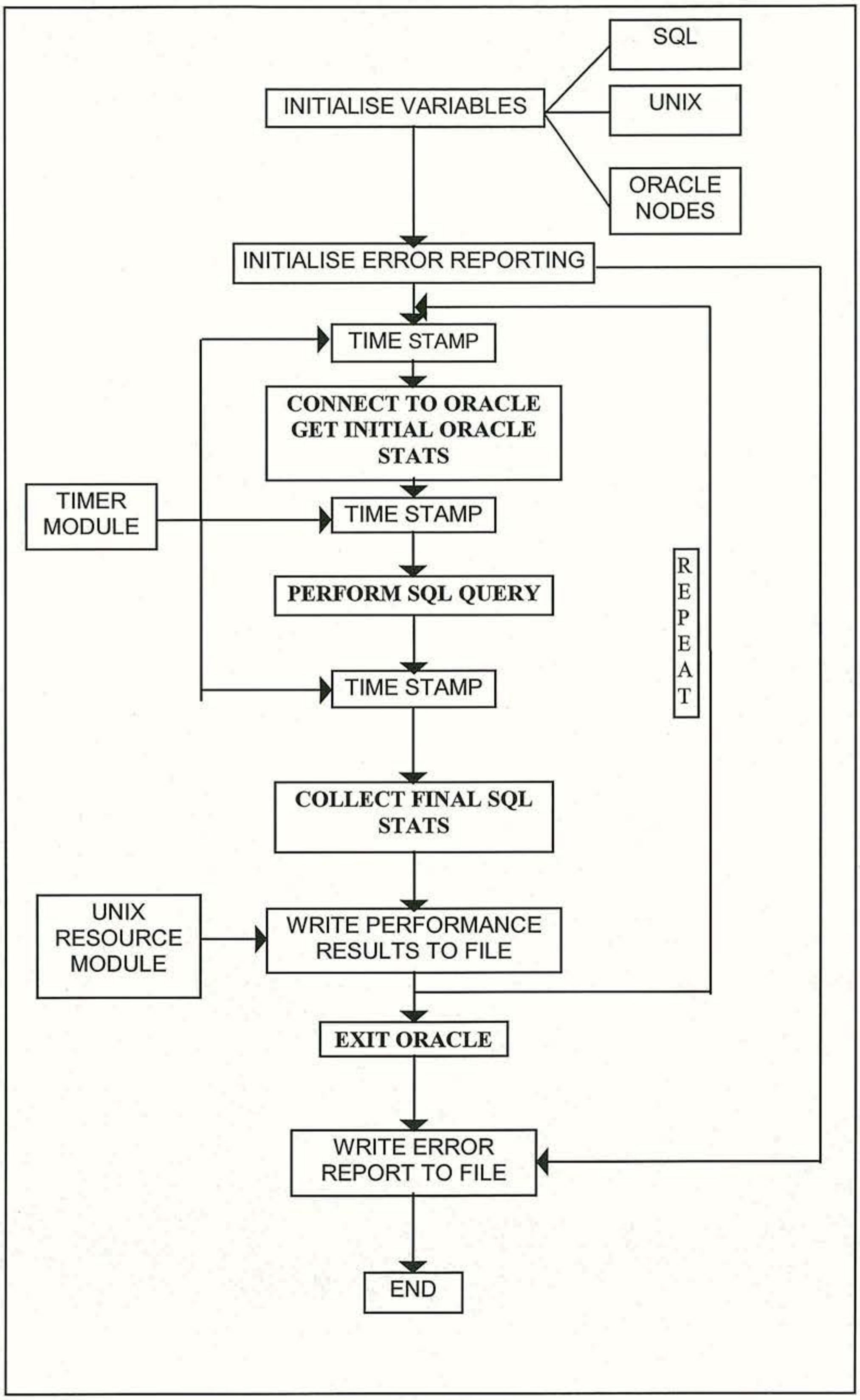


Figure 10-1: Performance Program Flow Diagram

10.2.2 Timer Module

The timer module uses the time of day as time stamp. Various time stamps are taken during the running of the program and are used to calculate elapsed running time of the query and to measure the length of time the Oracle connection sequence takes to complete.

Time of day is calculated and converted into seconds. To calculate elapsed time two time stamps are taken, one at the beginning of the measuring period and one at the end, then subtracted to get the elapsed time value in seconds.

```
 double timer()
 {
 struct timeval tp;
 struct timezone tzp;
 int res;
 res = gettimeofday(&tp,&tzp);
 return (double) tp.tv_sec + ((double) tp.tv_usec) *
   0.0000001;
 }
m1_time = timer(); /* starting time for SQL query */
```

10.2.3 Oracle Data Statistics

The Oracle database statistics are stored in a pseudo-table which is constructed in memory at the time of database initialisation and called *V$SYSSTAT* (a full list of the contents of the *V$SYSSTAT* table can be found in Appendix F). The code extract below uses a cursor to extract db block gets, physical reads, consistent gets and physical writes. These statistics can be used to calculate the buffer cache hit ratio to compute the rate at which Oracle finds the data blocks it needs already in memory.

```
EXEC SQL DECLARE C1 CURSOR FOR
   SELECT NAME, VALUE FROM V$SYSSTAT A, V$STATNAME B
   WHERE A.STATISTIC# = B.STATISTIC#
   AND (A.STATISTIC# = 28
        OR A.STATISTIC# = 30
        OR A.STATISTIC# = 31
        OR A.STATISTIC# = 29);
```

10.2.4 Oracle Instance Initialisation

The initialisation of an Oracle instance occurs in three phases. Firstly, userid and password variables must be declared to Oracle. The variables contain information used to connect to individual oracle instances on specific nodes. The second phase requires the creation of an Oracle logon array to store all logon information in one place to allow the user to connect to multiple instances Oracle. The final phase is where the Oracle connection is established for each instance mentioned in the logon array.

10.2.4.1 Variable Declaration

The Oracle variable initialisation begins with SQL BEGIN DECLARE section. All variables used to store the User ID, password and values returned from the database must be declared at the very beginning.

```
EXEC SQL BEGIN DECLARE SECTION;
     VARCHAR uid[40], pwd[30], name[20];
     int end_value, start_value;
     char category[5], town[20];
EXEC SQL END DECLARE SECTION;
EXEC SQL INCLUDE SQLCA;
```

The variables declared here are:

- **`UID`** - user ID for the Oracle user
- **`pwd`** - password for the Oracle user.

Other variables declared hold data from Oracle systems and user tables:

- `start_value` - the value of particular statistics held in the Oracle system table `V$SYSSTAT` **before** the SQL query has been executed.
- `end_value` - the value of particular statistics held in the Oracle system table `V$SYSSTAT` **after** the SQL query has been executed.
- `category` and `town` - values from the Oracle database

10.2.4.2 Set up Oracle Logon Array

To begin multiple instances of Oracle each one must be specified as part of an array, giving the username e.g. `OPS$METTE`, password and the address e.g. `@t:hydra1:holly1` of the particular instance (see Table 10-1), the details are then read one at a time into the Oracle variables `uid` and `pwd` and used to initialise the Oracle instance e.g. `holly1` on the chosen node `hydra1`.

Table 10-1: Oracle Logon Array

Array	Username and Password	Instance Address
strcpy(node[0],	"UID/PWD	@t:hydra:holly");
strcpy(node[1],	"UID/PWD	@t:hydra1:holly1");
strcpy(node[2],	"UID/PWD	@t:hydra2:holly2");
strcpy(node[3],	"UID/PWD	@t:hydra3:holly3");
strcpy(node[4],	"UID/PWD	@t:hydra4:holly4");
strcpy(node[5],	"UID/PWD	@t:hydra5:holly5");
strcpy(node[6],	"UID/PWD	@t:hydra6:holly6");
strcpy(node[7],	"UID/PWD	@t:hydra7:holly7");
strcpy(node[8],	"UID/PWD	@t:hydra8:holly8");
strcpy(node[9],	"UID/PWD	@t:hydra9:holly9");

The number of instances initialised at any one time depends upon the individual test. Table 10-1 is an example of all nodes on Hydra being initialised for performance testing. However, by naming individual nodes, or smaller groups, the MCS can be partitioned. Several different performance tests could be run on their own partitioned nodes or other applications could operate using those nodes not utilised by the database.

10.2.4.3 Connect to Oracle

The following performs an Oracle connection to each of the nodes on HYDRA.

```
    EXEC SQL CONNECT :uid /*IDENTIFIED BY :pwd*/;
```

The benefit of using this method of starting Oracle instances is that they are all started as close together as possible and are executing the query as close to the same time as is possible to achieve. This puts the maximum amount of stress on any particular datablock for that particular run as all of the instances are trying to access it at the same time. This requires the parallel lock manager to synchronise controlled access to the data blocks. Another interesting side effect is the pattern of retrieval of datablocks from disk, which is explored later.

10.2.5 Performance Query Execution

Individual performance queries are executed through a cursor. Any query or group of queries could be inserted at this point for presentation to the database. Depending on the test in question, this may be presented to a single node, a group of nodes or the whole system. The extract of code below is taken from the test harness and shows a cursor C2 and the SQL query it represents.

```
/* Open cursor for SQL Query */
EXEC SQL DECLARE C2 CURSOR FOR
   SELECT CATEGORY, TOWN
     FROM B_1000, C_333
    WHERE B_1000.BUILDING# = C_333.BUILDING#
      AND B_1000.BUILDING# BETWEEN 1333 AND 1667;
```

10.2.6 Performance Data Storage

The statistics are sent to file at the end of every run. The following data are stored for analysis:

```
Node used
Start time for run
Starting time for SQL query
Starting time for fetching
Elapsed time to start time of SQL Query (data searching
Elapsed time from start of searching to start of
retrieval
Elapsed time for run
end_time
Total elapsed time
Unix system statistics
Oracle Statistics
```

The time stamps taken at various points throughout the run are used to monitor the burden of performance overheads that are placed on the system in addition to the running of the performance test.

Each performance test is run twice. This means that the first time the query is run it does so without the benefit of the buffer cache, and therefore all datablocks have to be fetched from disk. In the second run the data are already stored in the cache, and therefore retrieval time should be quicker. Comparison of the two results gives an indication of how well the buffer cache is operating and what performance benefit can be achieved.

10.3 GIS/Database Interface Performance Harness

This second test harness is designed to be run from within Arc/Info and is written using AML[9]. Instructions are passed to Oracle through the Database Integrator module that provides the functionality to connect to Oracle and join tables based in Oracle with geographic features in Arc/Info.

The test harness is a flexible piece of AML code that wraps around database queries issued from within Arc/Info, and provides the ability to gather performance statistics for specific types and combination of query. It follows a similar pattern to the previous harness written in C. The test harness performs a series of set up procedures which initialise a number of variables with data from the GIS and the Oracle database. Time stamps are collected at various points during the test run to calculate elapsed times for various functions, the performance test is then run and final statistics are gathered and written to file.

10.3.1 Connection to Oracle

Connection to Oracle was through the Database Integrator `connect` command. The command required the user to specify a data definition file that contained the address for individual instances of Oracle, the username and password. The data definition file was called during the set up phase. The logon details were stored in a separate file to reduce the time taken to make adjustments to the set-up procedure. It removed the need to wade through the test harness code every time an Oracle destination was altered.

The code fragment below is an example of information stored in the data definition file. The file specifies the address for an Oracle instance.

[9] Arc Macro Language

```
6.0 Oracle DBI Server
$ARCHOME/programs/oracle
@t:hydra1:holly1
```

The syntax of the connect command is as follows:

```
connect oracle1 OPS$METTE/password
```

10.3.2 Oracle Statistics

Oracle statistics were collected using dbmscursors. A cursor is a method of retrieving data a row at a time from a database. DBMS cursors operate in a similar fashion to those in Pro*C, requiring the user to declare, open and fetch the data from the database. The advantage of the cursor is that it allows the user to select data from Oracle using native SQL commands. Data from the V$SYSSTATS table are selected using this method, as can be seen from the program fragment below.

```
dbmscursor c2 declare oracle2 ~
   select value ~
   from v$sysstat a, v$statname b ~
   where a.statistic# = b.statistic# ~
   and (a.statistic# = 28 or a.statistic# = 29 or
  a.statistic# = 30)
dbmscursor c2 open

loop until all data retrieved

  dbmscursor c2 next
&end
dbmscursor c2 close
```

The collection of statistics generated by Oracle system tables is important to gauge what work the database is performing through the interfacing mechanism and how comparable the performance is with a similar query submitted directly to Oracle from outside Arc/Info.

10.3.3 Watch Files

A watch file is used to gather performance statistics from within Arc/Info while performance tests are running. The performance statistics from Arc/Info are essential to understanding how the GIS operates when performing connections to external databases, and locating bottlenecks or inefficiencies if they are occurring. When a watch file is turned on it collects all data in a named file until it is explicitly switched off again. The results are text based. The code below is an example of AML code used to control watch file operation.

```
/* set up watch file to gather statistics from arc using &pt
functions

&watch watch.log

/* begin performance testing

&echo &on
&pt &on
```

10.3.4 Query Execution

Performance queries are submitted to Oracle using native mode SQL from within the Database Integrator. The advantage of using this method over other methods like the relate environment, which can be used to execute SQL queries in an external database, is that the query is only submitted once to Oracle, rather than for every single row of data retrieved. Native mode queries from Arc/Info are expressed as follows:

```
reselect mains arcs ^test1 where pressure = 'L'
```

where `test1` is the name of the relate, indicated by ^. The command means that arcs representing gas mains are reselected based on the gas pressure they carry. The arcs are held in Arc/Info and the pressure information in Oracle.

The relate itself is defined and stored in Arc/Info and defines the connection between Arc/Info and Oracle, specifying the name of the table identifier for spatial table, the database name, the table identifier for the oracle table, the type of access, and the name of the relate.

This type of access to the database requires tables and views to be optimised in the database rather than specifying all requirements from the GIS end. It also limits the number of relational joins that can be used. However, linking spatial attributes with a view created in Oracle rather than individual tables means that multi relational joins can be performed successfully.

10.3.5 Performance Data Storage

The contents of the watch file log and the data from the Oracle system tables were written to file for further analysis.

10.4 Constraints and Limitations

There were a number of constraints placed on collecting performance statistics from the parallel environment. The first major difficulty was that there was no monitoring facility that allowed the user to look at the performance of multiple nodes within the parallel machine. The usual monitoring tools were available on individual nodes, but could only collect information for that particular node. This made gauging the performance of the whole machine difficult.

Due to the configuration of the parallel machine, resources were used at a number of different levels, for example, the SPARC nodes, the transputer nodes and the parallel disk controller (cache). There were however no tools for monitoring performance below the level of the SPARC nodes. The extensive performance monitoring tools provided for monitoring the Oracle database went some way towards solving the problem. However, they could show what was happening to Oracle, but not to other applications.

It is also important to take into account that monitoring facilities have a small overhead when collecting and analysing performance statistics.

10.5 Processing Results

To run the performance tests at specific times of the day or night, a series of Unix command files were created. The performance statistics from each individual run were stored in files, which were organised into directories. Each directory represented an individual test run and was labelled with both date and time.

Once the performance statistics had been collected and written to file they were transferred to a spread sheet for further analysis.

SECTION 4

Performance Tests and Results

11. Phase I Tests

In this chapter the Phase I tests and results are presented. Details of the hardware, software, the construction of the pilot database and the test harness can be found in chapters seven and eight.

The testing falls into a number of stages which will be detailed in this chapter. The initial stages consist of verifying that both the hardware and software are all in working order, and that all of the interfaces between the different applications are able to provide the communication links that are expected of them. Effort has been concentrated on the MCS and Oracle v6.2 because little was known about this combination. A few simple tests were run to verify that Oracle was up and running on all of the nodes at the outset.

The second level of the tests detailed is used to verify the behaviour of the parallel RDBMS in the parallel environment. The version of Oracle v6.2 for the MCS, which was newly released at the time in question, was supplied with the most minimal of benchmarking information and it was necessary to perform a series of calibrated tests upon the database in order to compare its behaviour with that of serial RDBMS.

The final tests in Phase I begin to assess the performance of GIS applications using parallel RDBMS to store both attribute data and also spatial data. As part of this series of tests it was necessary to devise methods of submitting queries to the parallel RDBMS through GIS applications to specific nodes or groups of nodes on the MCS, and collecting and assessing performance statistics from the GIS, the RDBMS and the MCS.

To perform the tests in Phase I a small database was developed in conjunction with a series of calibrated SQL queries to examine:

- how processing is divided across the MCS
- the processing speed of a single node vs. nodes working in parallel
- the bandwidth of the network and the volume of the network traffic
- the ordering of joins

- the effect of the cache.

The results from this phase have several functions to perform in Phase II. The performance monitoring harness will be used to monitor and collect results, the suite of performance tests that are felt to be most indicative of parallel performance will be used on a much larger database, and the performance statistics collected will be used to compare the performance of the 'pilot' database with one of a size commensurate with those used daily in real life corporate situations.

11.1 Software And Hardware Tests

The first and most important series of tests was to establish the performance of the nodes on the MCS. The main processing nodes consisted of 10 SPARC processors linked together by a very fast network (CSN). The intention was to use groups of the nodes in parallel, and therefore, it was important to determine that they were all performing at a similar speed. The tests were looking for defective processors and possible problems occurring with the CSN. It was expected that malfunctions would initially manifest themselves by extended elapsed times.

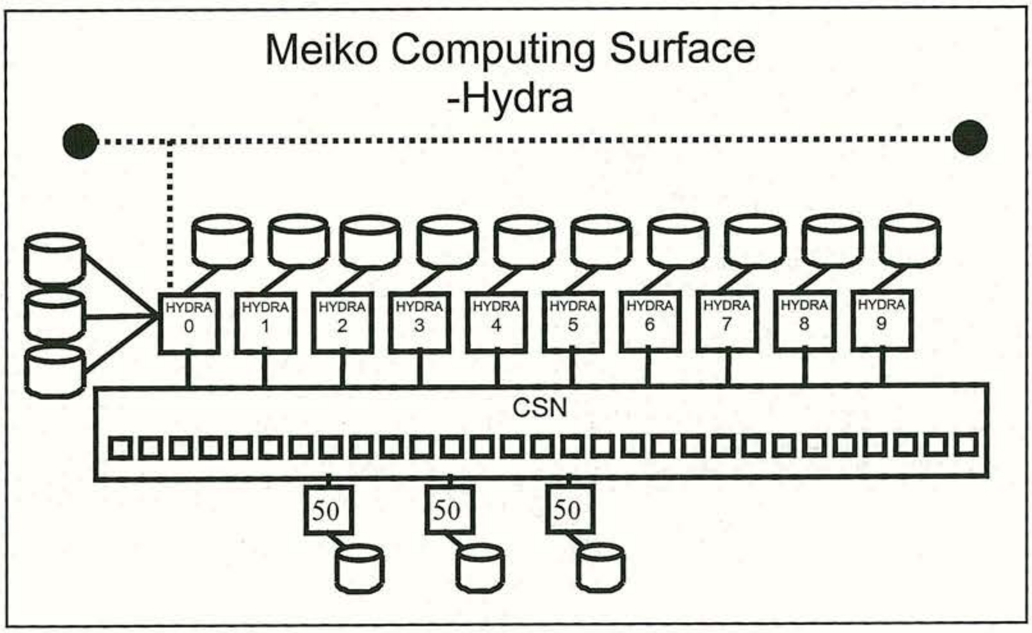


Figure 11-1: the Meiko Computing Surface known as Hydra

Initial sets of tests were run to establish the performance of the individual nodes on Hydra. The tests were run at a period when the system could reasonably be expected to be busy (2.45pm) and at a time when the system should be quiet (2am)[10].

11.1.1 SPARC Node Performance Tests

Three SQL queries were used for this performance test. The queries returned a specified number of rows, using a single relational join to select data. The queries used were as follows:

- 1a1_c
- 1a10_c
- 1a15_c[11]

which returned 111, 1111 and 1667 rows respectively. These were run with the aim of checking that the performance of each individual node remained similar to that of the other nine SPARC nodes in the system, as the number of rows returned increased.

Each individual query was run twice in succession to collect figures with and without the effects of caching. This was repeated five times and the results for each particular query were averaged out. For this test the total elapsed time (time taken for the SQL query to run twice), the system time, user time and the memory usage were examined.

11.1.1.1 Results

There are two elements of interest in these tests, the performance of the individual nodes, and the effect of traffic across the university campus network on response time. Most tests were done in batch mode late at night or early in the morning, when it was expected that the network would carry minimal traffic. However, some tests requiring a more interactive approach had to be run during the day. The parallel computing system was located in a building at some distance from where the author was based, and accessed using the campus

[10] It should be noted that there were certain parts of the system such as the main FDDI (optical connection) over which the author had no control. It can therefore only be assumed that these times would provide the types of conditions required.

[11] see Appendix A for SQL code for individual queries

network. It was therefore important to establish whether the traffic on the network was having an influencing effect on overall performance.

It can be seen from Figure 11-2 that all of the nodes excluding Hydra0 had a similar elapsed time performance for any given query submitted to Oracle. It could, therefore, be extrapolated, excepting Hydra0 which will be considered next, that the computer was working as intended with no single node showing either decreased or increased performance compared to its neighbours. Although, it must be noted that the level of work in this particular test was minimal compared to the work expected of such a machine, and will fit into the memory. It could also be observed that there was little difference in elapsed time between those runs performed at a potentially busy time of day i.e. early afternoon, and a quiet time of day i.e. late night/early morning.

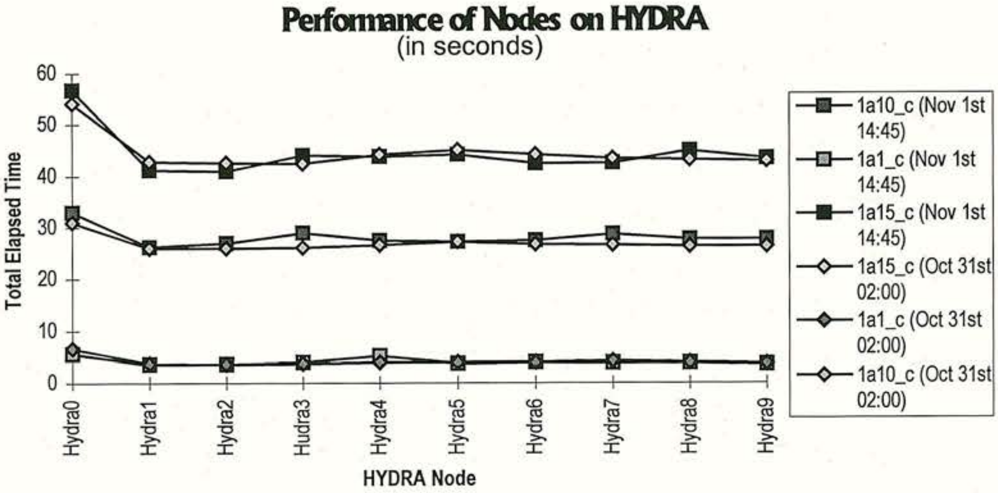


Figure 11-2: Elapsed Time Performance of Nodes on Hydra

The figures show that the effect of the university network had little or no influence upon the elapsed time of queries to the Oracle database, and therefore the effect of traffic across the LAN was negligible and will be discounted for the moment.

Although this is a very simple test of the nodes, it proved to be a good measure of the health of both Oracle and the MCS. Comparing the performance of the nodes at frequent intervals quickly highlighted any problems which would affect the response time, and hence the performance, of a particular run of performance tests. When using a parallel machine such as Hydra in a commercial situation, where the parallel GIS environment will be in constant use it is important to be able to identify hardware and software faults or bottlenecks quickly, particularly as parallel speed-up is often handicapped by the slowest performing node or task.

This test was used to gauge the state of the MCS throughout the project and proved to be an accurate barometer of the health of the computer, as can be seen from Figure 11-3. Figure 11-3 shows the response of the system 5 months on from Figure 11-2 and it is obvious that there were problems with both Hydra4 and Hydra7 which became more pronounced as the work load increased. There is also a potential problem with Hydra5, which may be as a result if Hydra4 is configured such that it uses the same inter-connect paths as Hydra5. It may also be an indication that there are problems developing with Hydra5 that need to be investigated.

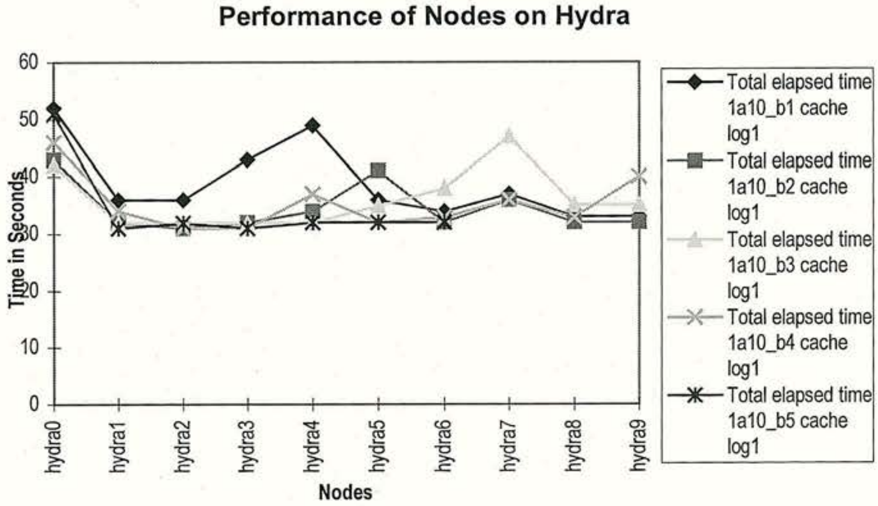


Figure 11-3: Performance of Nodes showing difficulties

As can be seen from Figure 11-2 there were some serious problems developing with this experimental system. Difficulties developed that caused the processing of Oracle to hang indefinitely, with no way to free up the system. All methods provided by Oracle for killing Oracle instances ceased to function when the system got into this state and there was no other solution that to abort Oracle by rebooting the entire parallel system. Unfortunately, although a huge amount of time was spent trying to solve the problem, it proved very difficult to trace. There were some hardware faults reported by diagnostic tests (data corruption and time-outs with the switching chips); however, it was very unclear whether these had any bearing on the observed problems with Oracle. The hardware faults eventually halted the testing project, a number of months earlier than was intended. The problems are explained in more detail at the end of the chapter.

11.1.1.2 Node Hydra0

`Hydra0` is the odd node out in the parallel system. For any given query run against the Oracle database it consistently recorded a longer elapsed time, regardless of whether caching was in operation or not. As the level of work increased the difference between the elapsed time on `Hydra0` and the other nodes (`Hydra1-9`) was proportionately longer (see Figure 11-4).

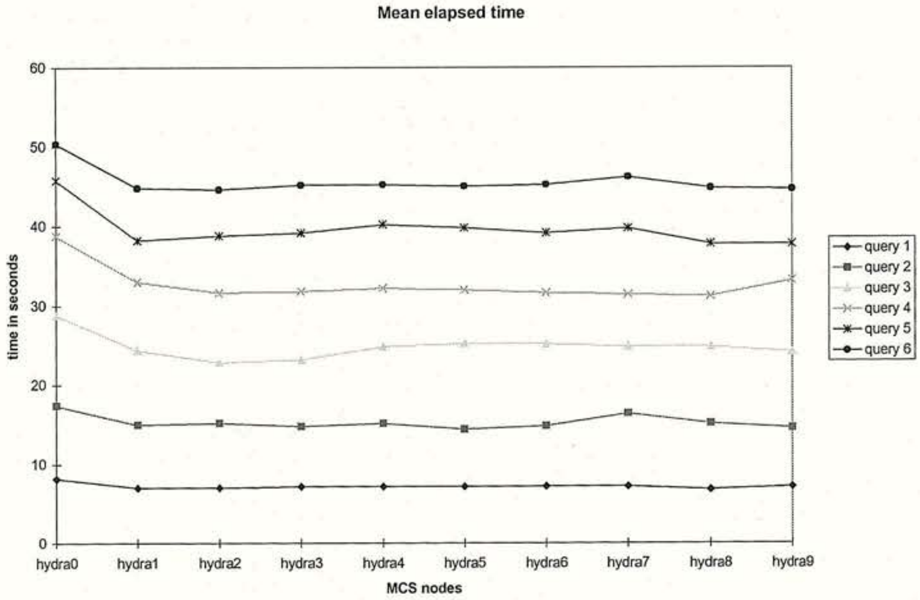


Figure 11-4: Mean Elapsed time for Hydra0 and the Other Nodes

Further, examination of other performance indicators such as the *system* and *user* time[12], and memory usage during the query runs brought other anomalies to the surface associated with `Hydra0` (see Figure 11-5, Figure 11-6, Figure 11-7). It can be seen that the values for these

[12] system time = percentage of CPU used by system resources.
user time = percentage of CPU used by users.

particular indicators are consistently lower than those of the other nodes, even though the elapsed time is always greater.

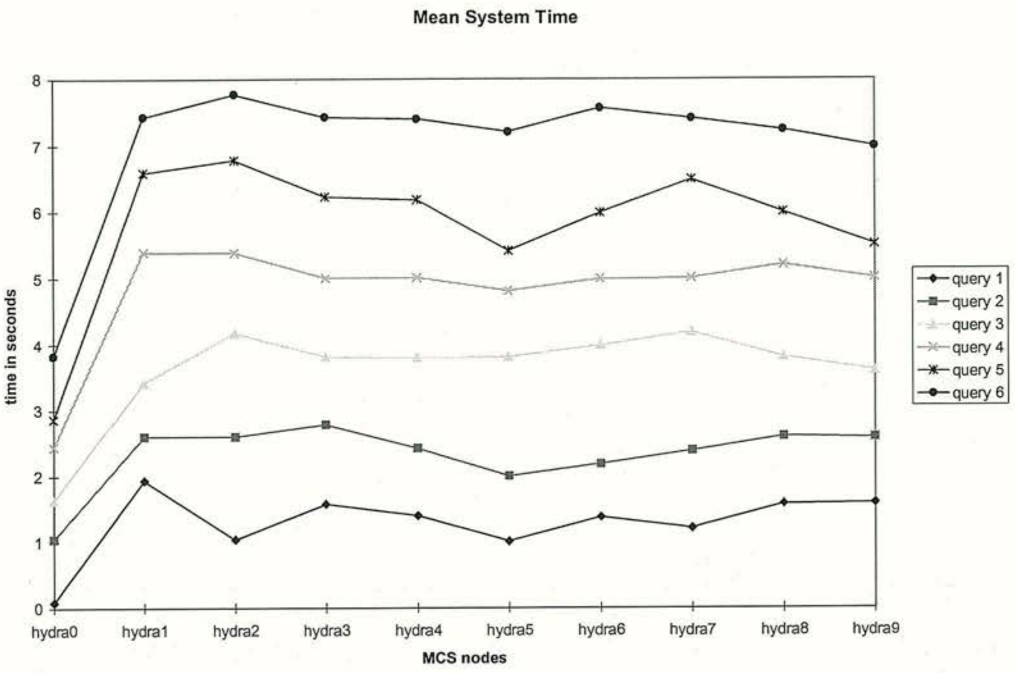


Figure 11-5: Mean System Time

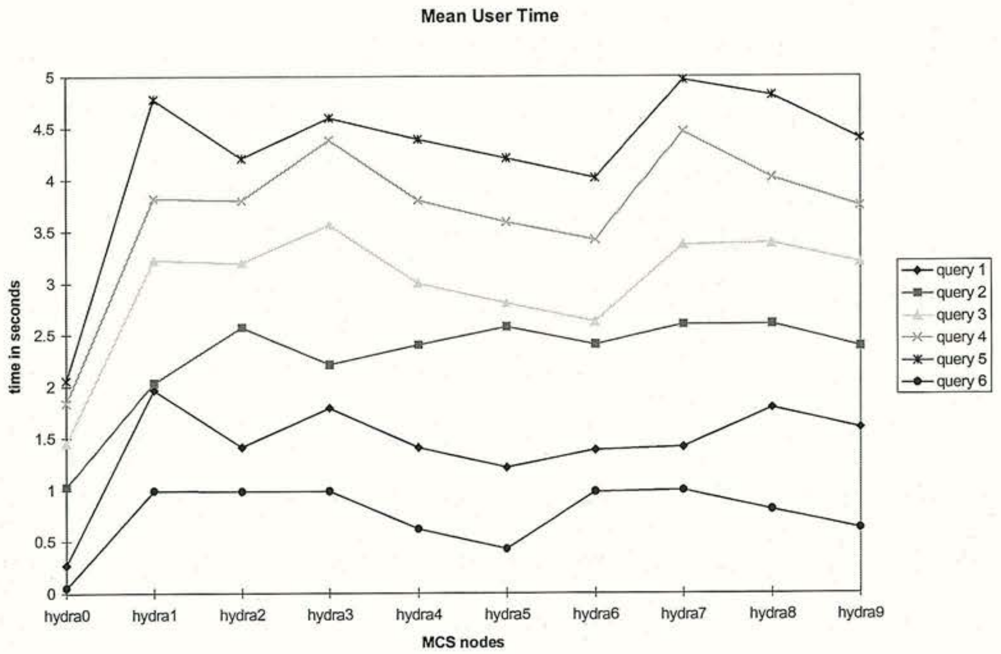


Figure 11-6: Mean User Time

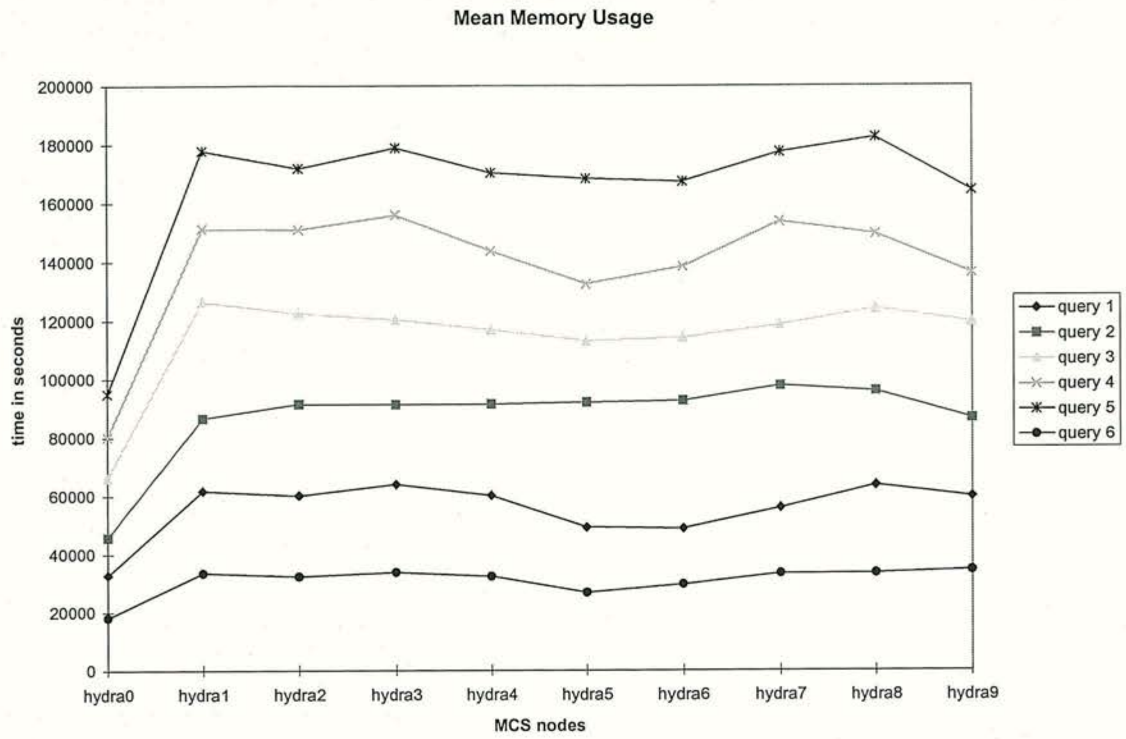


Figure 11-7: Mean memory usage for Hydra0 and the other nodes

The most likely explanation for this odd pattern of behaviour is due to the configuration of Hydra itself. The machine was configured such that the node `Hydra0` played the role of the server node and was therefore responsible for all of the external communication of the system. All work to the parallel computer required the use of external communication as the MCS was located in a separate part of the university campus and connected through the campus LAN using a mixture of FDDI link and ethernet. The consequence of this type of configuration is that there is a cost to handling external communication and it is necessary to consider the various options for handling it.

When running queries in parallel, consideration must be given to the fact that the node designated to be the server node will not display the same performance characteristics as the other nodes, it is likely to perform more slowly and thus reduce the overall performance of the system. Depending on the corporate requirements for the parallel database system there are two strategies that could be followed. The first is to designate one particular node as the server node and use it solely for that purpose, e.g. handling all communications, and use the other nodes to run Oracle. The other option would be to configure the system such that the overheads incurred by the server node are spread across the system. This may result in a very slight overall reduction in performance, or the overheads may be swallowed up by the other processing tasks and have no visible effect on performance. It is not in the remit of this research project to explore the effects of different hardware configurations. However, decisions about external communication support does merit further investigation in the future.

The task of handling external communication clearly had an adverse effect on the elapsed time for SQL queries. However, it was much less clear why the system, user time and memory usage were much lower for `Hydra0` over the other nodes (see Figure 11-5, Figure 11-6, and Figure 11-7). No immediate explanation could be found for this phenomenon. However it would be worth further investigation to see what bearing it has on performance.

11.1.2 Using Multiple Nodes To Perform Relational Joins

The database system is designed to maximise the number of users using a single database at any one time while maintaining the integrity of the system, security, and above all retrieval speed for each user. When a user logs onto the database there are three basic methods used to assign a particular node to them for the creation of the SGA and background processes. Two of these methods are automatic and the final is manual. The system can automatically assign a user to the least used node, or use a *round robin* method of assigning the user to the next available node on the list, which appears to be the default method, or the user can explicitly state which node to use as part of the logging on procedure. This ability to address individual nodes explicitly can be used to direct queries to a particular node, but it can also be used to split up the processing of a query so that separate tables will be processed on separate nodes.

Research into the distribution of data in a distributed database system (see Figure 11-8) showed that, when processing queries that involved tables held in different databases, for the majority of the time the most efficient method required the bulk of the processing to occur on the local node (shown as 'L' in the node sequence on Figure 11-8) to reduce communication overheads. However, research has also revealed that the processing power of the individual system was of importance, and in some instances where the bulk of processing fell to a more powerful remote node it was more efficient to carry out the processing on the remote node (shown as 'R' on Figure 11-8) and send the results back to the local processing node, as can be seen from permutation 8 where Hydra was acting as the remote node in the distributed database system (Chan, 1993).

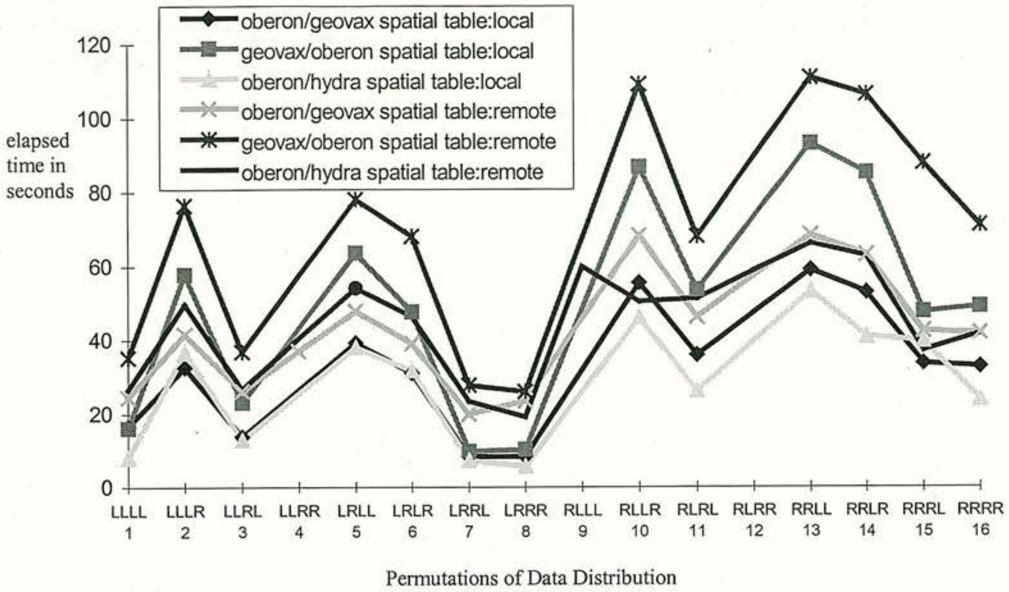


Figure 11-8: A Graph Showing Performance Behaviour of Data Distribution Using Different Local and Remote Nodes (as Chan, 1993)

A similar pattern can be seen when the individual nodes on Hydra are treated as nodes in a distributed database system (see Figure 11-9). If tables, used in data retrieval tasks, are accessed from different nodes on the MCS the two most successful strategies limit the communication overheads by ensuring most of the processing occurs either on the local node or the remote.

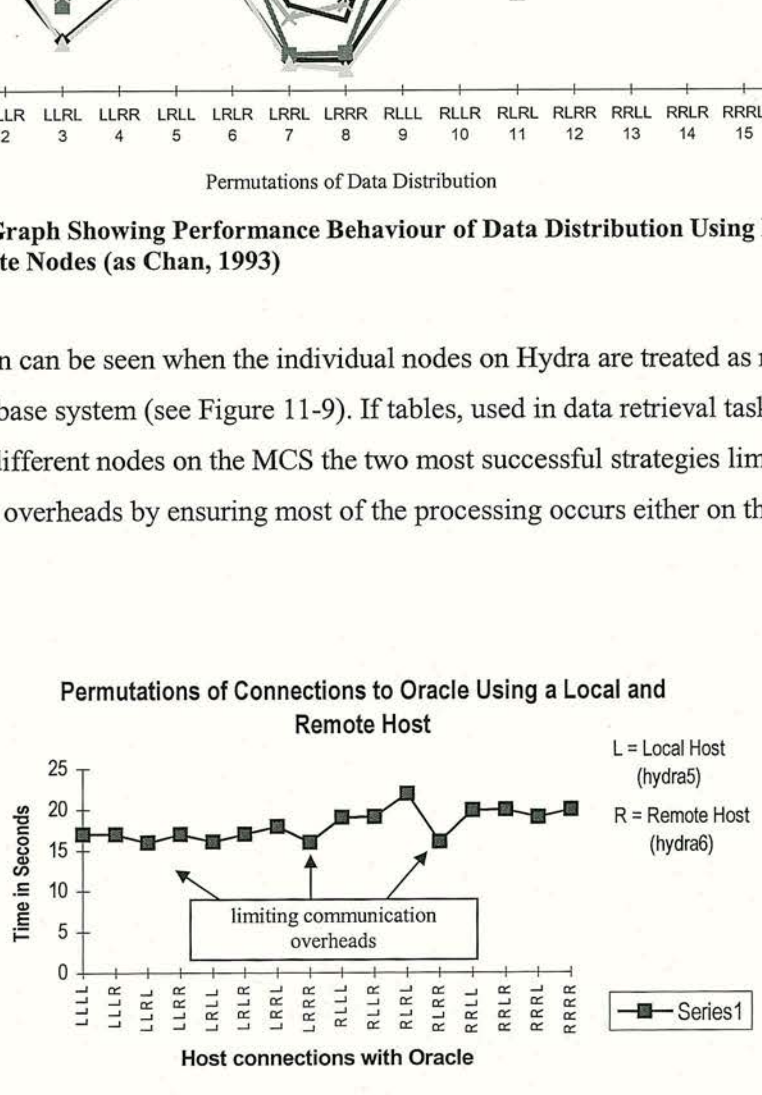


Figure 11-9: A Graph Showing the Performance Behaviour of Data Distribution Using Two Nodes of Hydra as Local and Remote Nodes

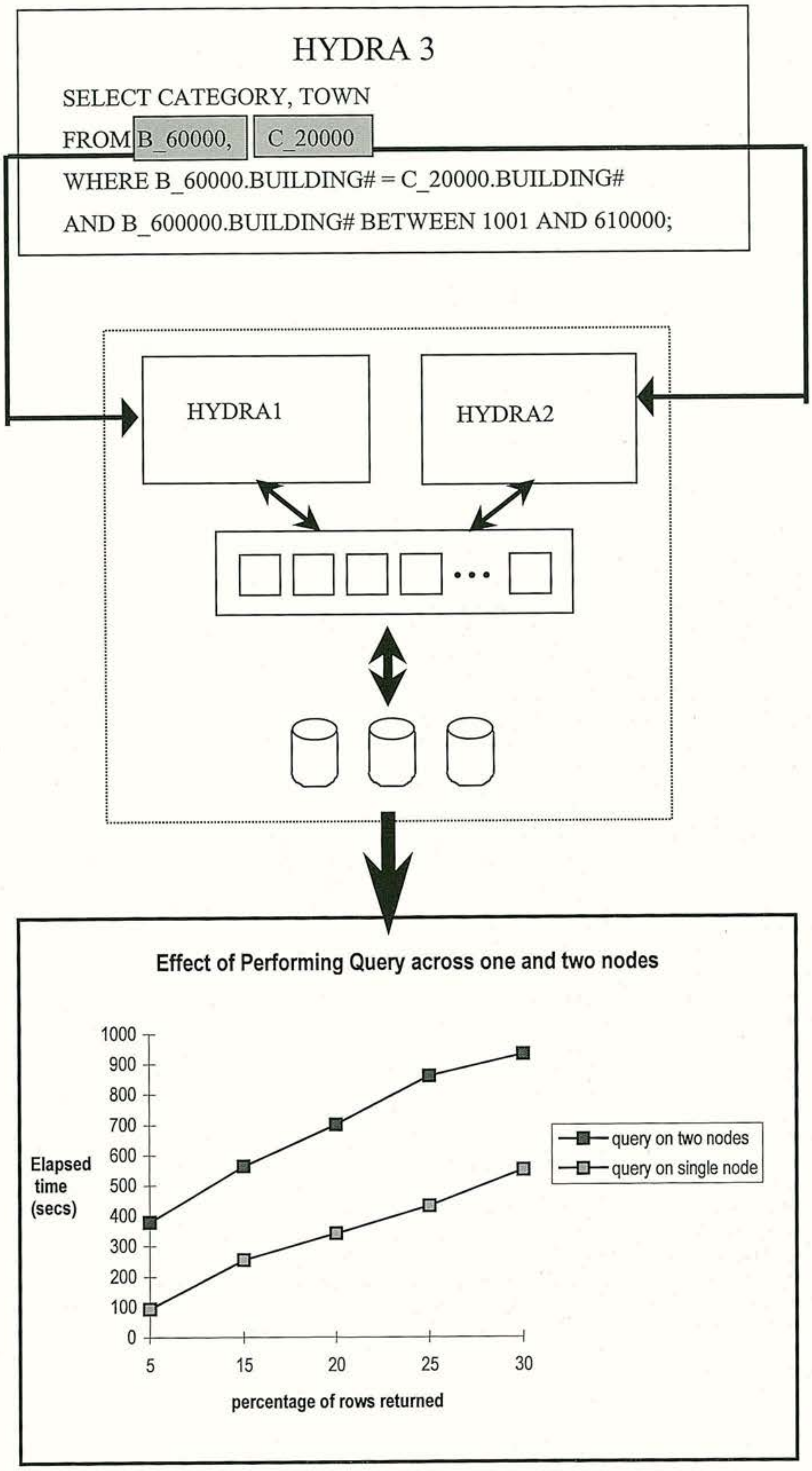
Although parallel query optimisation and execution does not happen automatically in Oracle v6.2, given the processing power of the nodes on Hydra, the high bandwidth connection (CSN) between the individual nodes and the specialised background processes, it might be advantageous to divide the processing of complex queries requiring multiple relational joins over several nodes. The advantages would be to spread the cost of processing so that no single node is working absolutely flat out, and spread the cost of communication overheads, to reduce the chances of bottlenecks occurring, and improve the speed of transactions.

11.1.2.1 Performance Test

Using the tables B_60000 and C_20000, and the query 1a_60b30 - a query with a single relational join, the query was submitted to the database. The query was run firstly on a single node, and then run using three nodes (see Figure 11-10) - the node initiating the query, and two further nodes, one for each table of the relational join. A percentage of the total number of rows in the table were retrieved ranging from 5%-30% of the table and the results plotted. The technique of splitting the processing of the query across two nodes was fairly crude, in that the two tables in the query were given different node addresses in the FROM section of the SQL query.

The result are graphs with similar patterns of increasing elapsed time as the percentage of data returned increases. However when compared with the same set of queries run on a single node it can be seen that any advantages that may have been gained from the ability to use separate nodes to process the individual tables in parallel are lost. The elapsed time for the query run across three nodes took almost four times as long as using a single node.

There are several possible explanations for a query spread across several nodes taking longer than that run on a single node. The first reason could be a communication bottleneck due to excessive message passing between the nodes involved. The second cause could be due to the SQL optimiser that is not set up to perform parallel query decomposition and may therefore take much longer to process the optimum retrieval path for the query. Finally, the indexing system has not been optimised for parallel query access, therefore there may be contention for use of indexes between nodes, or a full table scan has had to take place on each node. These three factors are explored in more depth below.



```
SELECT CATEGORY, TOWN
FROM B_60000, C_20000
WHERE B_60000.BUILDING# = C_20000.BUILDING#
AND B_600000.BUILDING# BETWEEN 1001 AND 610000;
```

Figure 11-10: Performance Difference between SQL query on One Node and Two Nodes

Communication

The first consideration is that there is excessive communication required between the nodes themselves, and between the nodes and the parallel disk farm that far outweighs the actual processing time. The size of the table used in the query is relatively small with only 60,000 rows and the complexity of the query is small so therefore may not create enough work to overcome the communications overhead. The advantage of this strategy may become apparent when using very large tables where the communication overheads form a small proportion of the overall time taken to process the query.

The next stage in this series of tests would be to try running the query on increasingly larger sized tables to examine whether increasing levels of work load would render the communications overhead negligible, or if, as work load increased substantially, so do the communications required to support a query spread over several nodes. It would also be useful to determine whether there is a point where the elapsed time for the query is less when the query is split over several nodes than on a single node.

Another cause for long response times that can not be identified with any obvious bottleneck, could be caused by *excess latency*. This is where the system topology in the file is defined such that the obvious path from the user processor to the instance processor exists but is not used, or the path exists, and is defined, but is also used for heavy file server traffic. In this case it would be necessary to reconfigure the MCS such that paths between defined user processors and instance processors exist and are not used for other types of communication (Oracle, 1991).

The intention of the author was to repeat the tests on tables developed in Phase II; however a combination of hardware and software failure made this impossible.

SQL Optimiser

The second influencing factor is that of the SQL optimiser. The optimiser is part of the Oracle kernel which examines SQL statements and chooses an optimal execution plan for that particular statement. The execution is a sequence of physical steps the RDBMS must take in order to perform the operation. When working out an execution plan the optimiser examines the following:

- details of the syntax specified
- any conditions the data must satisfy
- the database tables the statement will access
- any indexes that can be used to retrieve data.

Based on the information the optimiser works out the optimal retrieval path for the SQL statement being executed.

Using EXPLAIN PLAN, an optimiser diagnostic statement, the decisions used by the optimiser can be examined. In Oracle v6.2 the rule based optimiser uses a pre-defined set of precedence rules and indexes to sort out which path will be used to access the database. However, it is unable to make decisions based on cost, for example, if an index exists on a particular table the precedence rules **dictate** that the index be used. The optimiser is unable to make a decision as to whether the use of the index improves or degrades the performance of the query execution.

The SQL optimiser includes the ability to instruct the optimiser manually as to how the query should be executed by including *hints* in the SQL statement. Using this method it may be possible to improve the performance of the SQL query execution.

To plan an optimum parallel execution there are two specific areas to consider. The first is the FROM clause where the ordering of tables can be crucial if the rule based optimiser can not make an intelligent decision, in which case Oracle executes the query as it is written. The parser processes tables reading from right to left and, therefore the last table name in the FROM clause is the first to be processed. It is also advisable that the driving table (i.e. the last named table) is the smallest as this reduces the number of database blocks that need to be read. When designing a parallel execution plan it is necessary to consider how best to divide the tables up between the available nodes. It may prove beneficial to group tables together, with each group having its own driving table, and then combining the results of each node using a similar principle. The node with the smallest number of rows retrieved becomes the driving node.

The second consideration is designing an efficient WHERE clause sequence. The way the conditions of a WHERE clause are specified have a significant impact on the performance of SQL in version 6.2. In the absence of any other information the Oracle optimiser uses the conditions in the WHERE clause to determine the best retrieval path for the database. By specifying the most efficient clauses early in the WHERE clause the rule-based optimiser will be more effective in selecting the most efficient path. When working in parallel the ordering of the WHERE clause may affect the best method for ordering tables.

Finally, it should be considered what functions are used with the query when considering how to decompose the query. For example, grouping functions, such as GROUP BY, by their very nature require all of the data to be present before they can work, so queries containing these types of function will not perform well in parallel.

However, no matter how well the queries are decomposed using manual instructions to the SQL optimiser, or how carefully tables are ordered or the WHERE clauses constructed, the Oracle V6 optimiser is unable to *divide and conquer* database access. As a result, the time taken to retrieval data will always be poorer than that of databases using optimisers with that capability. In versions of Oracle using SQL v7 and later this ability has been added. Access using the divide and conquer method has the advantage that it allows database access without changing application logic, thus allowing it to break down queries and perform them in parallel.

Indexing

Data are accessed in two ways, either using a full table scan or using indexes. For most situations the use of indexes will greatly improve the performance of a query because they allow more immediate access to the selected rows. They are particularly effective for large tables where the overhead of using an index forms a very small proportion of the overall execution time (Corrigan & Gurry, 1993).

However, when more than approximately 25% of records in the queried tables are selected the use of an index becomes a disadvantage. Using indexes in this situation only increases the overheads because the database must perform a full table scan in addition to reading the index (Oracle Corporation, 1990). In a benchmark test by Corrigan & Gurry (1993) it was found that a row could be inserted into the EMP[13] table in less than 0.11 seconds. After

[13] a table supplied by Oracle

adding 8 indexes to the table (which is more than usually recommended for any one table) the time taken to insert a single row increased to 0.94 seconds, more than 8 times the original overhead.

This point may be of great significance when trying to decompose individual queries across a number of nodes on the system. The use of an index on the table B_600000 on building# attribute may magnify the influence of the index overhead when splitting the execution of the query between several nodes, because the overhead is incurred on several nodes. One method that may improve the performance of a single query split over several nodes is to store the tables and their indexes on separate disks - this will allow greater throughput (Oracle Corporation, 1991).

11.1.3 The significance of the cache buffer in multi-instance Oracle

The buffer cache is an area in memory that holds copies of database blocks for tables, indexes, rollback segments, and clusters. Each buffer holds one Oracle data block, the more blocks that can be held in memory, the less I/O is required, and the better performance will be (Corrigan & Gurry, 1993). By increasing the parameter DB_BLOCK_BUFFERS in *init.ora* it is possible to improve performance by up to 50% for long running update jobs and between 5-20% for transaction processing.

Oracle v6.2 has parallel cache management which uses instance locks to co-ordinate access to database blocks held in buffer caches. Each individual instance of Oracle has its own SGA (System Global Area) which includes the buffer cache. The parallel cache manager co-ordinates the access of individual database blocks for each instance. If the required blocks are not held in the cache then they are located from disk and put into the cache. The cache is held on a Least Recently Used (LRU) basis. If a request is made for a sequence of database blocks not held in the cache and there is no free space the oldest entries in the list are removed and replaced by the newest. A cache view supplied with Oracle contains information about the contents of the buffer cache (Oracle Corporation, 1991).

The size of individual buffer caches on Hydra was 16 mbytes and required a table of 4 columns, each of 23 characters long of more that 160,000 rows in order to flush it. The efficiency of the cache can have a dramatic effect on the database performance and requires monitoring on a regular basis. Therefore the following test was devised to measure the effect of the buffer cache.

11.1.3.1 Performance Test

When designing the performance test harness for gathering performance statistics it was decided to run each individual query twice in order to collect values for both the uncached run where database blocks had to be brought into the cache and a cached run where the database blocks were being held in memory. Therefore, the test consisted of running a *flush* query to clear the buffer of current data blocks held, then run individual queries twice in quick succession. The first time the query is run, the required datablocks will have to fetched from disk, the second time, the block will be resident in the buffer cache. The effect of the cache can be seen in Figure 11-11 where the same queries are run twice on each node. The first time the query is run the results must be fetched from disk, the second time they are retrieved from memory making the elapsed time of the query shorter.

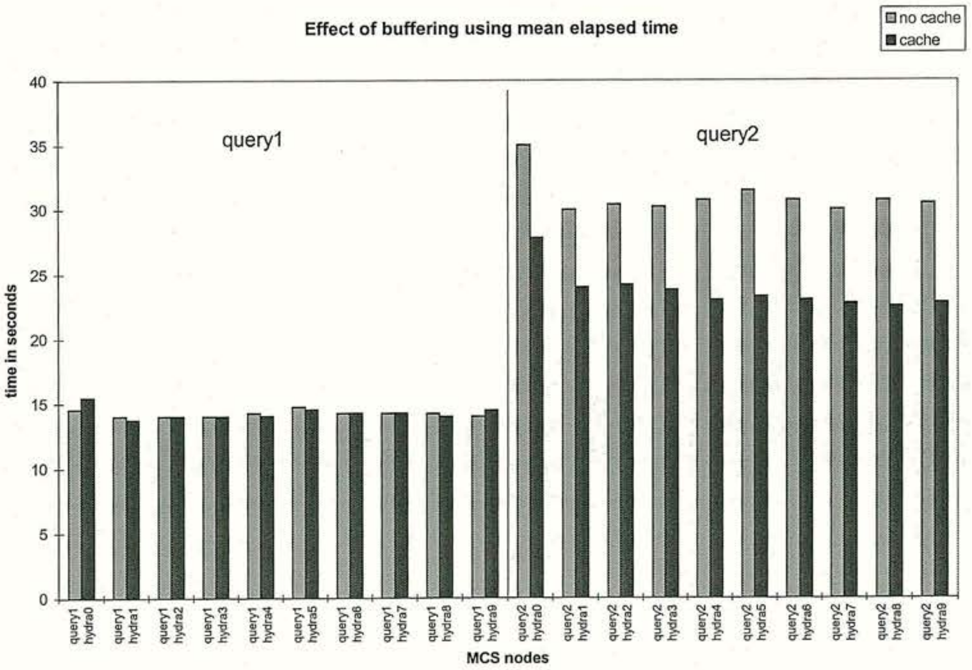


Figure 11-11: Effect of Caching on Elapsed Time

A similar effect can be seen when system time is examined (see Figure 11-12). For the most part system time is shorter when data is retrieved from the cache rather than from disk.

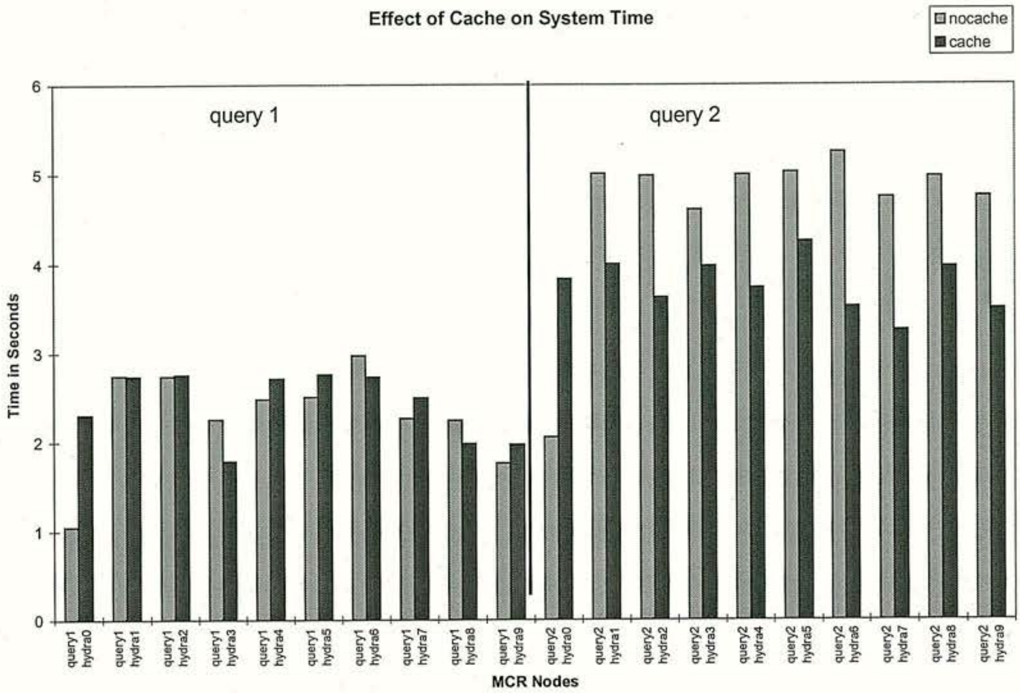


Figure 11-12: Effect of Cache on System Time

From a series of tests done on data loading the following examples have been used. The graphs show average elapsed time and average system time for two queries.

	query name	rows returned
Query 1	1a10_b	1111
Query 2	1a20_b	2222

For query 1a10_b the effect of caching appears to be negligible with an actual decrease in average performance of 0.04% (elapsed time). For query 1a20_b the effect of caching was an average improvement in performance of 24% (elapsed time).

11.1.3.2 Cache Hit Ratio

One measure of the effectiveness of the buffer cache is the cache hit ratio. When an Oracle process accesses data stored in a cache the data are read directly from memory without the overhead of disk I/O and is known as a *cache hit*. The size of the cache affects the likelihood of a request for data resulting in a cache hit. The cache hit ratio measures the proportion of total data requests satisfied entirely by memory access rather than disk I/O (Oracle Corporation, 1991).

There are several methods for gathering the statistics to measure the cache hit ratio. Two scripts called BSTAT and ESTAT are supplied by Oracle to allow the user to take snapshots of how the database is performing. The scripts have been found to incur very little overhead (Corrigan & Gurry, 1993). The system table *V$SYSSTAT* (see Appendix F) also stores the system wide value for each statistic in the table *V$SESSTAT*. *V$SESSTAT* is the system table that holds the current value for each statistic held by Oracle. The statistics can be used to interpret how the database system is performing and for tuning various parameters to improve performance if necessary. During this phase of the project statistics from these tables were gathered and used to calibrate the buffer cache, applying the calculation

hit ratio = (logical reads - physical reads) / (logical reads)

where *logical reads = consistent gets + db block gets*.

As a rule of thumb if the hit ratio is below 0.9 then the system needs some tuning (Corrigan & Gurry, 1993).

The cache hit ratio was unable to be measured for Hydra because the figures collected from the table *V$SYSSTAT* through the performance harness were very unreliable and without meaning. Several possible causes for these are detailed below. These include the effect of multi-processing, and the parallel disk farm cache.

Multiple processing

During the statistics gathering exercise many of the tests required that multiple processes were running on Hydra. Often all 10 nodes were accessing data simultaneously. When the Oracle statistics were gathered for each individual run the system was in a state of flux due to the processing occurring simultaneously on the other nodes. Therefore, the figures that were recorded were continually being updated and would appear to be meaningless in terms of statistics gathered for individual nodes. This would seem to be a drawback of only being able to monitor individual nodes rather than the whole parallel machine. It may have been possible to use figures from the table *V$SESSTAT* but hardware and software problems became manifest before this was done.

Parallel Disk Farm Cache

On the MCS the performance measurement of cache efficiency is complicated by the parallel disk farm which has a cache of its own. If an Oracle instance retrieves a series of database blocks from disk they are deposited in the disk buffer as well as in the instance buffer cache. When another instance requires those blocks, it first checks the buffer cache, then the disk cache and finally the disk itself.

In order to propagate the queries across all of the nodes on Hydra it was necessary to create a master file in UNIX that submitted each of the queries in turn to the database. The MASTER file was started manually on each node with as little time between the start of the MASTER file on the first and last nodes as was possible. Therefore, the pattern of retrieval of database blocks for each query will be complex. For example, given three instances running on separate nodes, individual instances requiring the same database blocks will first check their buffer cache, then the disk cache and finally the disks themselves. If **instance1** retrieves the first few blocks it will place them in the disk buffer making them accessible to the other two instances. The other instances access those database blocks and store them in their own cache. **Instance3** is the first one ready for the next database blocks in the series so it pulls the blocks from disk and deposits them in the disk buffer, which the other two nodes access. **Instance 2** then locates the next database blocks, **instance1** was delayed due to a slight bottleneck so **instance3** locates the next database blocks from disk ... and so it continues with the instances leap-frogging over each other to retrieve datablocks from disk.

The effect of the disk cache on performance of the buffer cache is difficult to measure and may be responsible for some unexpected effects. An example of this is where the first run of a query which was theoretically measured without the benefit of caching had a shorter elapsed time than the second run, which should have taken advantage of caching. This effect can be seen in Figure 11-13 where the query that should have made use of the cache on the nodes `hydra4` and `hydra5` are clearly slower than the first run. The effect can also be seen clearly in Figure 11-14 where the results of same query is shown having been run at different times of the day, and also on different days. It can been from the graph that when the query was run and recorded in log3 there was a significant drop in performance between the query when it was run for the first time without the benefit of caching, and the second time when caching should have taken effect. The straight line on the graph indicates the percentage increase in speed-up of elapsed time between the first and second runs of the query.

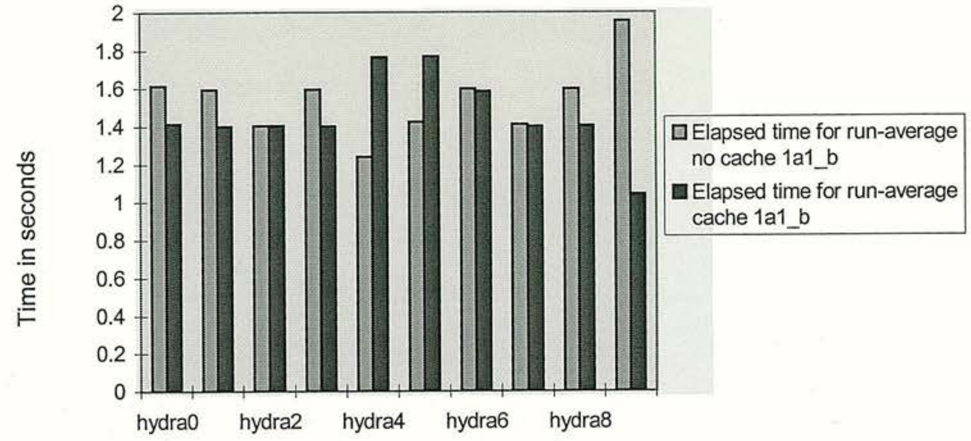


Figure 11-13: Cache contention and LRU list

11.1.3.3 Advantages of the cache structure on Hydra

The design of Oracle on Hydra where each individual instance of Oracle has its own separate buffer cache that is not shared with any other instance is very useful in a database system that is intended to support different types of transaction and different database applications because the computer can easily be partitioned with groups of nodes working on separate applications with no interference from other nodes. This means that one part of the machine could be configured to perform transaction processing, or updating procedures to the database while other nodes are dedicated for GIS use. Providing the machine configuration is such that the inter-connects paths do not coincide, the system response time should not degrade to the same degree as in a sequential system.

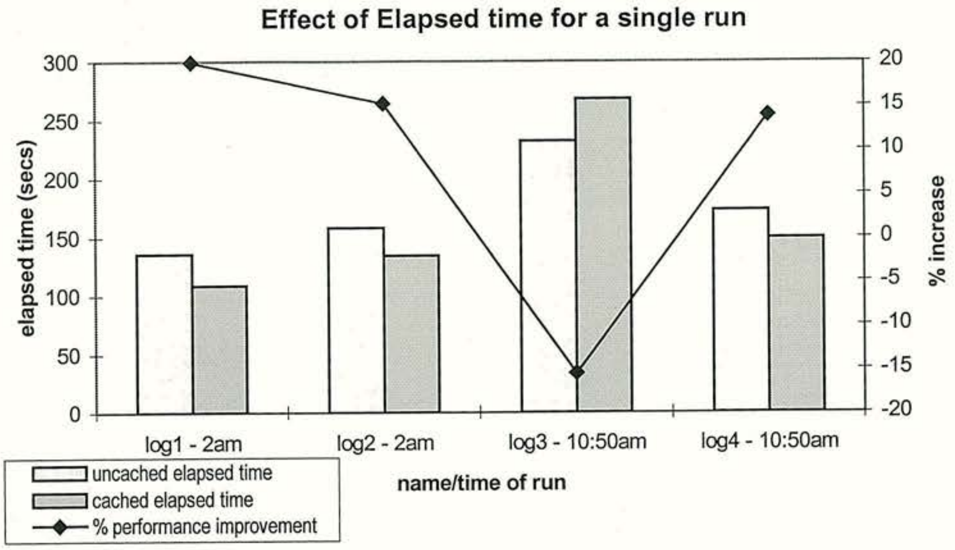


Figure 11-14: Percentage Performance Improvement Using Buffer Cache

11.1.4 Other Performance Tests

Two other sets of performance tests were devised for phase I testing, but due to a combination of difficulties experienced with the hardware and software these could not take place. The ideas for the two tests are explained below

11.1.4.1 Data Loading

The first of these tests was to examine the effect of data loading on the parallel database. Firstly using a single node of the computer: to test the effects on elapsed time as the number of rows returned was increased for a single user; then to increase the number of users as well as the number of rows returned. The aim of this test was to discover whether the performance of Oracle on the MCS would be adversely affected if one node was loaded significantly more than any of the others. This would be of particular interest to those wishing to partition nodes on the MCS for different functions or user groups. If one particular node or small group of nodes monopolised the processing power to the extent that other users were severely affected, it would be useful to know under what conditions this happened and whether there were any solutions.

It was expected that performance would degrade as the level of data increased, and would further degrade as the number of users increased. What was unclear was whether the rate of performance degradation would be linear, and whether a point would be reached where resources were swamped, causing a sharp decline in performance. If there was a point where performance dramatically declined, it would be interesting to note at what point the decline began and what was causing the bottleneck. It may well be possible to reduce the rate of performance decline by reconfiguring the MCS or retuning Oracle.

The second stage of data loading involved multiple nodes on the MCS and multiple users on each of those nodes. There were a number of permutations for these tests that involved different data loading on each node and multiple users, building up to a point where full capacity was reached.

The aim of this test was to establish whether it was possible to reach full capacity on the system and if so what the performance profile resembled. It was expected, given the design of the parallel database and Oracle and the descriptions of performance by Oracle, that if the level of data loading and number of users was spread fairly evenly over the MCS the performance would decline gently. However, it was not clear if this would be the case if part of the machine was more heavily loaded than the rest, and whether this would cause a sharp decline in performance. If performance under these conditions was adversely affected then a re-examination of the configuration of both MCS and Oracle would be necessary.

11.1.4.2 Effect Of Percentage Of Data Returned Using Indexed Tables

The other performance test that was curtailed by hardware and software failure was that of the effect of using indexed tables. For this a series of calibrated tests were designed using different percentages of data returned to measure any indexing effect.

The aim of these tests was to examine the effect of multiple users retrieving data from the same tables, as would be expected to happen if a database were accessible from the internet, to many thousands of users. It is known that indexing tables can have a beneficial effect on retrieval times under the right circumstances, usually when less than 25% of the table is being retrieved. However, if more than 25% of rows are returned then the index tends to add to the overheads and extends retrieval time rather than improving it. In an environment where thousands of users were accessing data from the same tables to varying degrees it would be difficult to predict what effect indexing would play on performance.

The effect of indexing was to be considered in several ways starting with the retrieval of 0, 20, 50, 80 and 100 percent of data from a single table on a single node. This would then be repeated using a single relational join query and multiplied up until the whole system was being used with multiple users.

It was expected that, as the percentage of retrieved data rose, performance would begin to decline more sharply. However, it is not clear what would happen in a multi-user situation utilising all nodes on Hydra. It should be possible to derive some rules of thumb for creating indexes in such situations, should the performance tests be run.

11.1.5 Hardware and Software Problems

During the lifetime of the research project a number of hardware and software difficulties occurred which ultimately brought the project to an early end. The main problems occurred when queries were executed on Oracle, run either in real time or in batch mode. Some fault caused Oracle v6.2 to become inactive, freezing on all ten nodes. The only way to resolve the problem was to reboot the MCS, a practice that was particularly unpopular with the other users of the system! It was unclear whether the fault was primarily a hardware one or a software one, but in either case it became insoluble. The MCS was lent for the project and Meiko were unwilling to spend a lot of time and energy looking for a fault on a machine that was soon to be dismantled. Oracle had moved to version 7, and although version 7 may have solved the problem there was not enough time within the project to begin again with a new database. Therefore the performance testing phase was halted and the results of some tests remain speculation.

12. Phase II Testing and Results

The following chapter presents the performance tests and results for phase II of the performance testing phase of this research. The thrust of this second stage of testing is aimed at the use of a GIS parallel environment in corporate situations. Therefore, the tests concentrate on the features that would have most influence over performance. Three of the most influential areas are the efficiency and flexibility of the GIS/database interface, the use of views to represent complex database queries, and the ability of the database system to manage a mixed work load of both long and short database transactions.

The success of both hybrid and integrated model corporate GIS in an integrated IT strategy, is heavily reliant on the performance of the interface between the GIS and the database. A powerful database able to store and manipulate large volumes of data, and a GIS optimised for processing and analysing large spatial datasets can not wholly compensate for an inefficient interfacing mechanism. The interface has an effect on both what the user sees, i.e. elapsed time for individual functions, and also on system resources, i.e. network traffic, CPU usage and disk access. The GIS/database interface is likely to be heavily used; therefore, it is important that the interface does not generate unnecessary work, particularly in an environment where thousands of users are accessing spatial and attribute information.

The use of views for operations involving spatial data stored on the GIS and attribute data on the database is very common. Views provide a useful shorthand for long and complex queries that are frequently used. However, they can also be used to circumvent the shortcomings of GIS/database interface mechanisms, particularly if there is no support for native SQL queries, if only a small set of SQL commands have been implemented, or if the interface does not support many-to-many relationships.

The ability to support mixed transactions in a single environment is also of importance if the parallel database environment is destined to form part of a data warehouse, support internet applications, or form the heart of an integrated IT policy. The reason is that data from many different sources will be brought together, and accessed in the same environment. Therefore, transaction processing operations will be competing for the same resources as the longer transactions of GIS and other MIS applications. It is also likely that groups of nodes on the parallel computer will be partitioned for different functions or usergroups, but all will be accessing data from the same database. It is important that the different types of transactions do not significantly impede the performance of each other and render the system unusable.

The tests from phase I performed a series of baseline performance tests on the database to characterise the performance of Oracle instances on individual nodes, and the performance of the database when all nodes were submitting SQL queries and retrieving data. The results from phase I showed that the nodes of Hydra all had similar performance patterns for Oracle processing, apart from `Hydra0`, and that the communication infrastructure was not impeding the performance of any node. Queries performed on `Hydra0` were found to take longer than its counterparts. This was thought to be because the node was responsible for handling communication to the outside world, and therefore, it was advisable to use this node for communication purposes only. The tests also showed that the buffer caches for the SPARC nodes and the disk controller had an effect on the performance of individual queries, although the effect of the cache was uncertain as it was difficult to measure the effect of individual caches. Nevertheless, the effect of the cache should be borne in mind when analysing the performance of other test queries. The results from the phase I testing were used to direct the phase II testing stage. For example, all subsequent tests undertaken excluded `Hydra0` from the testing schedule.

The following sections describe the tests and results on the performance of the Arc/Info GIS/ database interface, the effect of using views, and the effect of mixing different transaction strategies. Both the performance tests and the results are presented below.

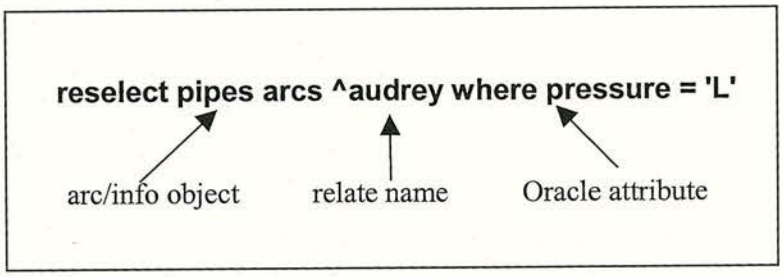
12.1 Interface Performance Tests

The investigation into interfacing issues considered the success of various strategies for retrieving data. The tests were designed for use with both integrated and hybrid GIS models, but due to operating difficulties explained in the previous chapter the tests concentrated purely on the hybrid model. The hybrid GIS model is still the most prevalent in the business world and it was felt the results would be of more interest at this stage. The data retrieval strategies detailed in the following pages cover the retrieval of data directly from tables held in Oracle, and using views (see Figure 8-2) to access the data.

The interface performance tests explore three different methods of accessing data using Arc/Info as the GIS application and Oracle as an external RDBMS. The first is a direct link from the spatial data held in Arc/Info to tables in Oracle. The other two methods involve using views to access the data.

12.1.1 Interfacing Arc/Info to Oracle

The first performance tests investigated the ability of Arc/Info to link to Oracle using the Database Integrator to establish and maintain the link. Spatial data, stored in Arc/Info was displayed on the screen using the ArcPlot module. Individual, or groups of gas mains were *reselected* using a *relate* (see Figure 12-1) to extract the appropriate attribute and MIS data that was specified by the relate definition file. The test harness developed for gathering performance statistic from within Arc/Info was used to gather the results.



```
reselect pipes arcs ^audrey where pressure = 'L'
```

Figure 12-1: Example of an Arc/Info relate

The first series of tests performed in phase II were designed to establish the performance characteristics of the different stages of data retrieval involved in the relate mechanism. For any given SQL query submitted from Arc/Info using the Database Integrator, three different stages of processing could be measured. These were the parse, execute and fetch stage. The parse phase measured the time it took for the SQL code generated by the relate environment to be passed to Oracle and prepared for execution. The execute stage performed the SQL query and the fetch stage retrieved the rows of data and passed them back to Arc/Info for display on the screen. An example of this can be seen in Figure 12-2 where the total elapsed time is displayed against the elapsed time for the individual stages of retrieval.

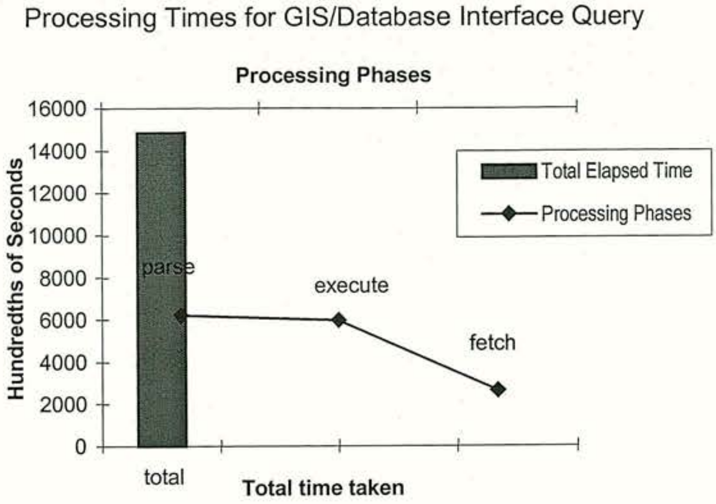


Figure 12-2: Processing time for an Interface Query

From the graph it can be seen that the parse and execute stages accounted for the greater part of the retrieval time, and that the parse and execute elapsed times were roughly similar in length. Figure 12-3 shows the proportion of time spent by the individual stages of retrieval.

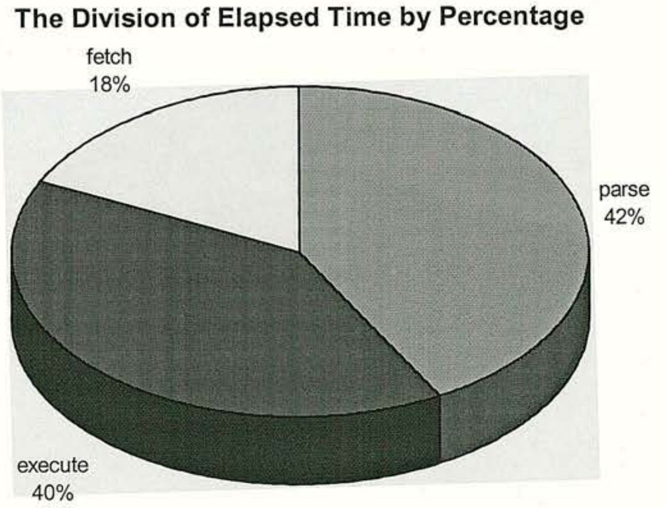


Figure 12-3: Proportion of Elapsed Time Spent on Individual Processing Phases

This indicated that effort in improving performance of GIS/database interfaces should concentrate on the parse and execute stages of the process. If the CPU usage for the same query is examined (Figure 12-4) it can be seen that the greatest processing effort was concentrated on the execute stage.

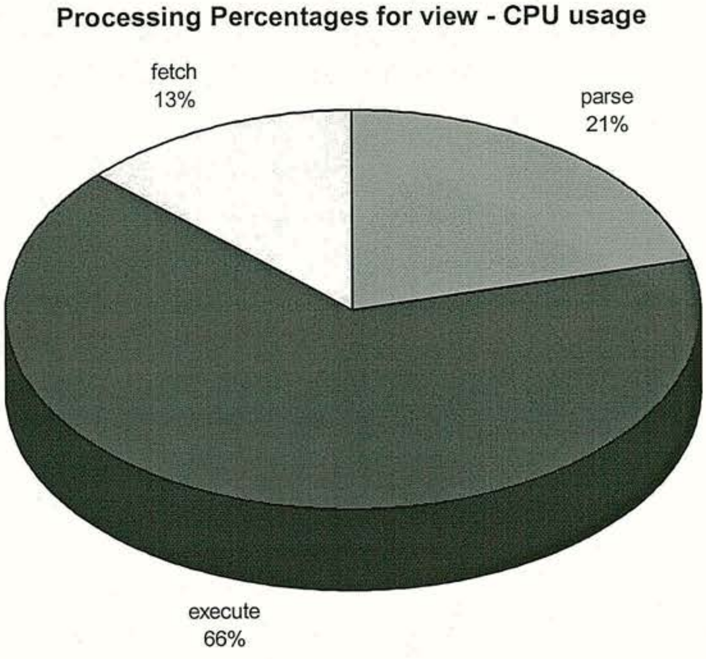


Figure 12-4: CPU Usage for GIS/Database Interface Queries

Two issues arise from these observations, the first on the execution stage, and the second on the parsing stage.

12.1.1.1 Query Execute

The execute part of the query took a proportionately longer time to run and also had a higher CPU requirement than the other two processing stages. Therefore, it is feasible that it was this part of the processing where bottlenecks were likely to occur first, particularly in a situation where multiple users were accessing data through the Database Integrator.

There were a number of alternatives available for investigating the effect of the execute stage on the performance of the parallel database and the MCS. The first strategy was to increase the workload by creating more complex queries (see the example of SQL code on page 218) to discover whether the time spent on the execute stage increased in a linear fashion as the level of work increased. The second was to increase the number of users utilising the GIS/database interface, and finally to combine the two strategies to increase the workload significantly for a worst case scenario. The advantage of performing the tests in this order were that it would identify if there was any difference in performance by increasing the work load through greater complexity of retrieval or shear volume of users. It would also identify whether there was a point where execute time increased rapidly, or whether a point would be reached where execute time levelled off. This would be useful for two reasons: the first would indicate the likely performance if multiple users were running similar queries in the same environment, and secondly it would indicate at what level performance would begin to decline sharply.

Once this was established a series of tests with calibrated workloads were established. However, hardware and software difficulties prevented the tests from going ahead. The following is a description of the test outlines.

Workload tests

The workload tests involve building up the number of relational joins in a particular query, and building up the number of rows of data that match the query. As the query became more selective the number of rows of data retrieved would inevitably decrease. Therefore, the volume of data in the tables accessed would have to increase as selectivity increases. An example of the most complex query built for the performance test can be found on page 221.

Multiple user tests

Multiple users of GIS on multiple nodes of Hydra would be set up and run simultaneously, starting with a single user on a single node and building up to multiple users on all nodes available. Performance statistics from the tests above could then be compared against performance values gathered those from the multiple user tests,

Result Conclusions

The aim of the two tests was to identify whether in fact the execute stage of query processing was causing bottlenecks in the system. If bottlenecks were evident then there would be two basic strategies to follow. The first would be to retune Oracle to improve the execute time. This would involve identifying which parts of the execute stage were the most resource intensive and time consuming. One candidate would be the query execution path the database used to retrieve the data. If the query is complex and the options available to the query optimiser are difficult to choose between then it may take a while to decide on the optimal retrieval path. The second would be to try implementing parallel query execution where the CPU costs are spread across several nodes. If this were followed up, a further test to investigate whether performance would improve would be necessary, using the same testing strategy (with multiple GIS users on multiple nodes, submitting queries to the database through the Database Integrator) as above but implementing parallel execution techniques.

12.1.1.2 Query Parsing

The parsing stage of query execution has a proportionately long elapsed time, but does not involve a particularly high level of CPU usage. It would, therefore, be useful to examine the effects of more complex queries on parse time to consider whether it is a variable that would have a significant influence as workload increased. It is likely that the solution to decreasing the time spent parsing SQL queries would be to redesign the database interface to reduce the amount of communication necessary and to make optimal use of the communication links available. It is possible that the communication path between two nodes on the MCS might influence parse time, particularly if that path were also used for other high traffic processes. Therefore, it would be beneficial to consider the configuration of the communication paths.

12.2 Views

There are a number of options available for constructing queries for use with GIS interfaces. The first option is to submit the query direct to the database using native SQL. The second is create a view (see example on page 221) of the data tables to be accessed in Oracle such that the request from Arc/Info to Oracle appears to be a simple table to table link. The third option is a halfway house, where a view is created on the data tables required in Oracle, but no WHERE clause specified. The WHERE clause can then be constructed as part of the SQL query from Arc/Info to Oracle. All three methods are viable and all three have their uses.

A series of tests were devised to investigate the performance of these three approaches. The results in Figure 12-5 and Figure 12-6 show the differences between the different strategies using both elapsed time and CPU usage. It is interesting to note that while the CPU usage for all three approaches is very similar, the elapsed time shows a significant increase in elapsed time for the strategy where the WHERE clause is defined outside the view.

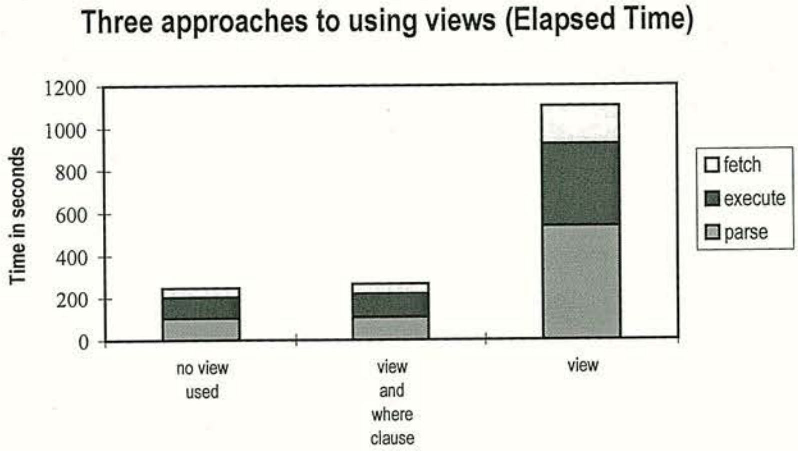


Figure 12-5: Graph to Show Three Approaches to Using Views - Elapsed Time

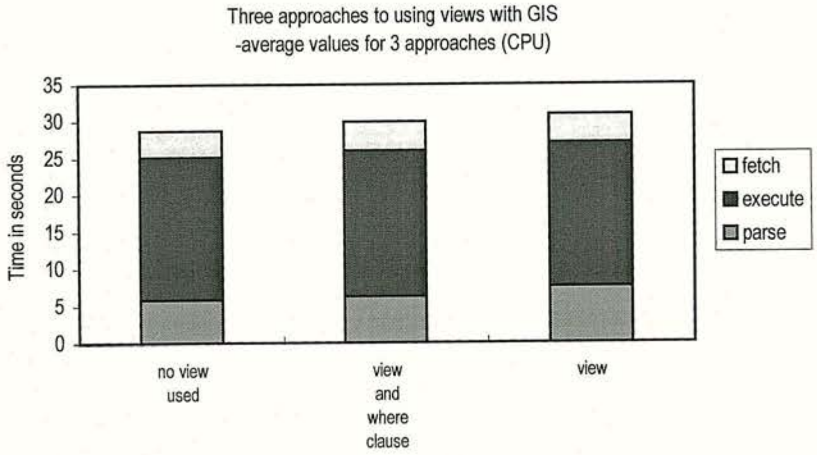


Figure 12-6: Graph to Show Three Approaches to Using Views from GIS - CPU Usage

The use of a view would always add a small overhead to retrieval time because it is necessary to process the view table before beginning to process the actual query. However, the effect of separating the WHERE clause from the view had a dramatic effect on retrieval time. The greatest effect appeared to be on the parse time. It is likely that a query such as that on the next page caused a large amount of communication to occur between Arc/Info and Oracle, and that query is demonstrating a serious weakness in the GIS/database interface.

These results require further investigation to identify precisely what is causing the huge increase in retrieval time. However, this was not possible to follow up due to hardware and software difficulties that were described in chapter eleven.

```
create view gas1
as
select meter_types.id meter_type_id, mete_id, char_id, tax_id,
hous_id,
counties.id county_id, town_id, stre_id, buil_id, cust_id,
       cust_pipe_record_id, pipe_main_id
from meter_types, meters, charges, households, customers,
council_tax, counties, towns, streets, temp_build, pipes
where pipe_main_id = cust_pipe_record_id
and buil_id = cust_build_id
and stre_id  = buil_street_id
and town_id = stre_town_id
and counties.id = town_county_id
and cust_build_id = hous_build_id
and hous_id = tax_household_id
and char_id = tax_band
and cust_id = mete_customer_id
and meter_types.id = mete_meter_t_id
and char_rate = 400
and meter_types.id = 3
```

Example of complex SQL Query used as a view within Oracle

12.3 Transaction Processing in a GIS/Database Environment

This final set of performance tests is devoted to the conflicting requirements of transaction processing vs. long, complex GIS queries. The aims of these tests are to examine the effect of both transaction processing queries and longer transaction queries running in the same environment.

A series of performance tests were devised to compare the running of a particular query: on its own; with a low level background transaction processing; with a high level of background transaction processing; and finally with a high level of transaction processing occurring on all nodes excluding the node the query is running. The SQL code on the next page is an example of a transaction processing query used in the performance tests. The transaction processing queries were supplied with enough data from the corporate database to run for at least an hour, to allow time for the start-up the SQL queries on all of the nodes.

```
select cust_id,a.mete_reading,b.mete_reading, b.mete_reading-
a.mete_reading
from meters a, meters b, customers
where ((cust_id = a.mete_id
       and a.mete_date between '01-JAN-93' and '31-MAR-93')
and (cust_id = b.mete_id
     and b.mete_date between '01-APR-93' and '30-JUN-93'))
```

Example of A Transaction Processing Query

There were some interesting results from these tests. Figure 12-7 shows that there is little effect when transaction processing is occurring on only one or two nodes on the parallel computer.

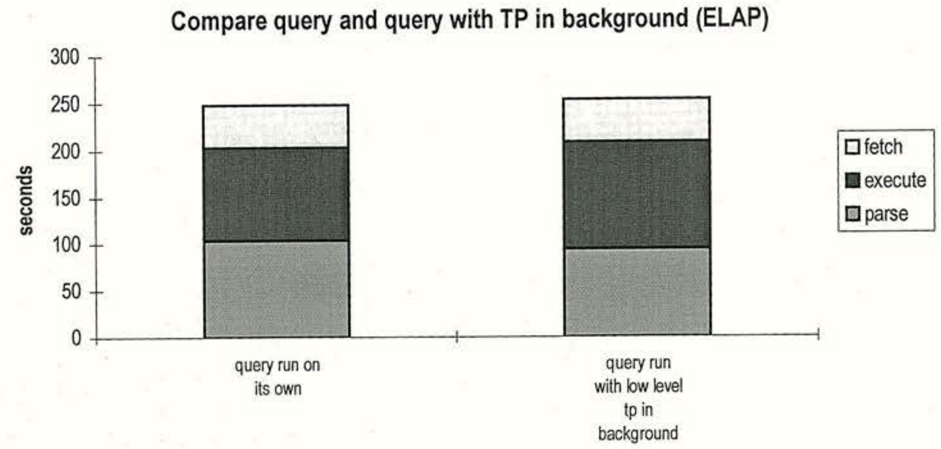


Figure 12-7: Graph to Show Effect of Transaction Processing

The effect is increased when transaction processing is occurring on all nodes on the Hydra, including the node that the complex query is running on. However, if transaction processing is excluded from that node, the effect of the transaction processing queries running in the background is negligible (see Figure 12-8).

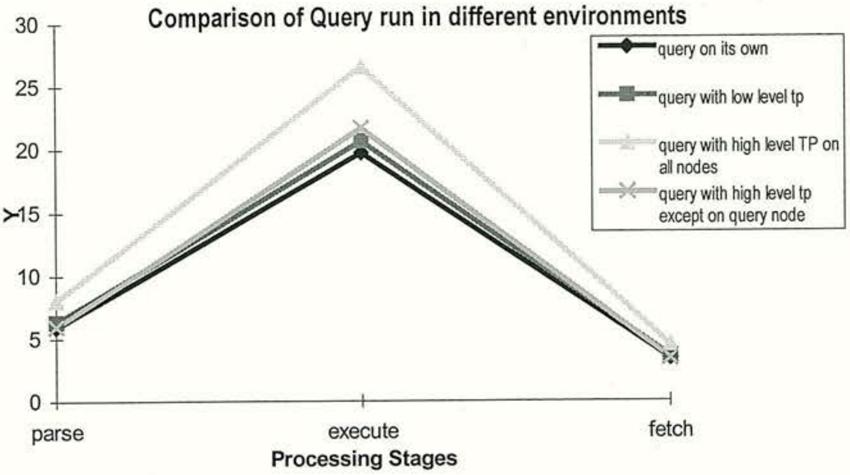


Figure 12-8: Processing Time for Queries Run in Different Environments

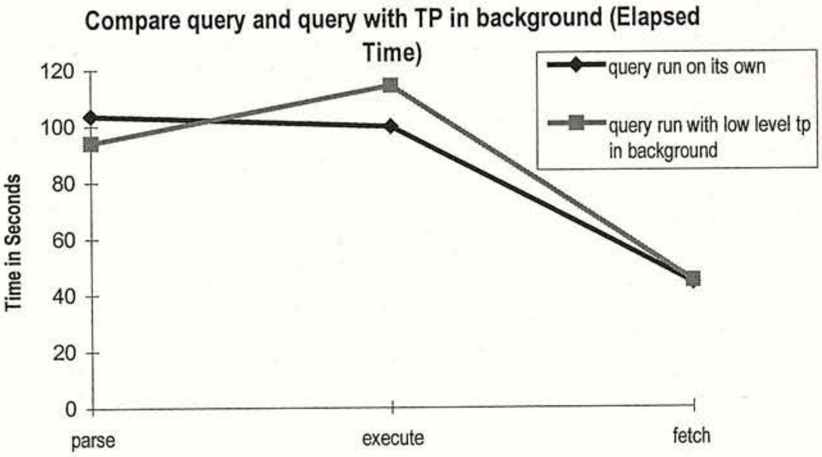


Figure 12-9: Comparison of Two Queries

One interesting feature to note in both Figure 12-8 and Figure 12-9 is that although the elapsed times (see Figure 12-10) are roughly similar, even when transaction processing is run on the same node as the complex query, it is the execute stage of the elapsed time where performance is most affected. Both the parse time and fetch time remains almost exactly the same.

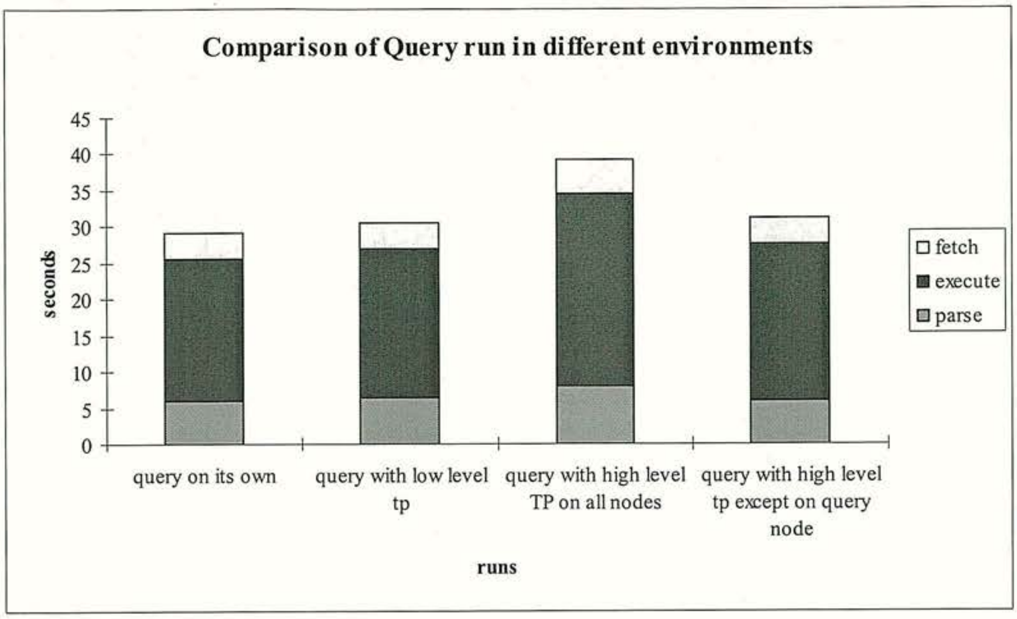


Figure 12-10: Comparison of a single query run in four different environments

12.4 Summary

The phase II performance tests described in this chapter have considered three fundamental features of a corporate GIS environment where the data store is located on a parallel database system.

The interface mechanism between GIS applications and the database is one area that can have a significant impact on the performance of the whole system. The interface mechanisms include both hardware and software components. The hardware components such as the communications network infra-structure of the parallel machine is well-known and has been the subject of much research, forming an important part of many commercial benchmarks. This is because it is such a fundamental part of the machine and a recognised bottleneck. The software components that form part of the GIS and also database software, are more of an unknown quantity. Many of the commercial GIS available were designed on the hybrid model, and while it was recognised that there was a need to make provision for external data access it was not a fundamental feature of the system and therefore not as efficient as it might be. Interfacing can be broken into three separate phases – parse, execute and fetch. Of these the parse phase, where the SQL query is passed to the Oracle database from the GIS, is processed for execution, took a significant proportion of the total retrieval time. Performance could be improved by making the link between the GIS and database much more streamlined to minimise the communication required. The execute phase of the query processing also required significant resources in particular CPU. One option to improve execute would be to retune the database, although it should be borne in mind that the database is designed for a multi-transaction environment, and changes to improve the execute phase of particular queries may prejudice the performance of others, for example, transaction type queries. Another option would be to implement parallel query execution.

The second area of interest considered whether the design of SQL queries from the GIS to the database has a detrimental affect on performance. Three options were examined: using native SQL, a hybrid query using a view of tables and a separate WHERE clause, and a view containing the whole query (stored in SQL). The results showed that while the first two options had a similar performance the use of a view to access data had quite a significant overhead. The most efficient method of interfacing between Arc/Info and the Oracle database was using native SQL commands issued directly from the GIS to the database.

The final area of consideration was the effect of mixed transactions in a single database. A series of tests were run where different levels of transaction processing occurred while more complex queries were being processed. Low level transaction processing had little effect even when the transaction processing and more complex queries were occurring simultaneously on the same node. High level transaction processing had some impact on performance, particularly if node0 (the communications node) was included as part of the test. If node0 was excluded transaction processing had a negligible effect on response time even when complex queries were run simultaneously.

13. Conclusion and Discussion

13.1 Performance Test Summary – Phase I and Phase II

The main aim of the research project was to investigate the integration of GIS and corporate database management systems in a parallel environment. To this end a series of performance tests were devised to explore the ability of the parallel system to function under different work load conditions, support multiple users (hundreds or even thousands are anticipated with the expansion of the internet), and provide an efficient interface between GIS applications and the parallel database. The performance tests were split into two phases, phase I and phase II.

Phase I was designed to measure the performance of the parallel database alone. The performance of the database under different workloads and user levels was measured for two reasons. The first reason was to provide an understanding of how the database worked and performed. The software was newly released with virtually no available benchmarking information. The second reason was to provide a base line of information for the phase II results. Phase I used synthetically produced data to allow a precise definition of workload during testing. The testing was broken down into six subsections, three of which were conducted on a single node, and three that used multiple nodes, from two to ten depending on the nature of the test. The test concentrated on the effects of relational joins, indexing and data distribution on the database system.

Phase II was designed to represent a more realistic use of a corporate parallel GIS/database system where the data held in the database were accessed by GIS through an interface mechanism. The data model designed for the purpose was based on a utility company and used a more realistic inter-linked dataset, designed to represent the day-to-day business transactions of the company. The performance tests were designed to test a number of data retrieval strategies that may be adopted when using hybrid or integrated GIS models. Of particular interest was the response of the system to running both on-line transaction processing type queries and highly complex queries simultaneously within the same database to gauge whether a more open corporate data access strategy could be supported using parallel technology and software.

Two performance harnesses were developed to run the performance tests and gather the performance statistics generated each time the performance test was run. The results were gathered, analysed and presented in this research.

Due to a serious fault with the parallel GIS/database system the tests were brought to an early close. However, the results gathered before the system became unusable are summarised below. The fault, which lay either in the parallel hardware or database software, caused the database to continually crash and render the whole database unusable, to the extent that the only method of recovery available was to reboot the parallel computer. The difficulty with resolving the fault was twofold. Firstly, the parallel computer was lent for the project and due to be broken down into its separate components and returned shortly, so there was little interest in locating a hardware fault. Secondly, the version of Oracle used in the research project, although newly released was soon superseded, so there was little interest in locating a software fault in an older version of the software. It was concluded that a move to a later version of Oracle was not a viable option due to time constraints and the project halted earlier than originally intended.

13.1.1 Results

13.1.1.1 Phase I

The results from Phase I were divided into four different parts. The first set of results described the running of the parallel machine. Performance tests were run on each individual node in turn to measure and compare the response of individual nodes against each other. From these tests it could be seen that all ten nodes on the MCS were of comparable performance except for node 0. Node 0 was the node responsible for handling external communications, and it appeared that the extra communications occurring placed a significant overhead on CPU usage. A simple test based on this performance test was devised to assess the health of the parallel computer by comparing the performance of each node against the other nine. Any anomalies in performance showed up clearly, indicating which nodes were affected.

The second stage of phase I was to use multiple nodes to perform relational joins. Two methods were explored. The first method simulated the data retrieval techniques of a distributed database by notionally assigning particular tables to particular nodes. This strategy worked most efficiently when all of the communication occurred on either the local node or the remote node, and took longest when communication was forced to occur on two nodes simultaneously. The second method was to attempt to parallelise a query using 2 or more nodes to retrieve the data. The results of this experiment were not encouraging. Elapsed time for the query took far longer when the query was divided up by this method than when run on a single node. A combination of excessive communication, an inability to optimise the query and a lack of parallel indexing brought this line of enquiry to an early close.

The third area of research considered the influence of the buffer cache. The results for this set of performance tests were unexpected. Under normal circumstances a query run against a database is likely to run much slower when submitted for the first time and speed-up in subsequent runs. The reduction in elapsed time on the second and subsequent runs occurs because the data blocks required to satisfy the conditions of the query are already stored in the buffer cache. Retrieval of data from the buffer cache is much quicker than retrieval from disk, therefore it is usual for the second run of a given query to take less time. However, this did not always prove to be the case when running tests on the parallel database. On a number of occasions the elapsed times for two consecutive runs of an individual query were either of a similar length or the second run had a greater elapsed time than the first. Due to the lack of performance monitoring tools for certain areas of the MCS such as the parallel disk farm and associated buffer cache, the reasons for this were difficult to investigate. However, it may be due to the fact that all of the nodes on the MCS access the same buffer cache. Therefore, the sequence of data blocks read from memory, used and then stored in the buffer cache could be unpredictable when more than one node is accessing the same table simultaneously.

The final area for research was the effect of data loading, where the response of the database would be measured as workload and user load increased. This set of performance tests were abandoned due to hardware and software difficulties described above.

The results from the performance tests in phase I were used to calibrate the data model in phase II.

13.1.1.2 Phase II

The phase II tests were split into three parts and the results are as follows. The first set of tests concentrated on the performance of the interface existing between Arc/Info and the Oracle database. Three different strategies were devised to access data from the database to display on the screen using Arc/Info. These involved using native SQL commands, creating a series of views in Oracle and finally using the Arc/Info database integrator to retrieve data. Of the three methods of retrieving data from Oracle, native SQL commands passed directly from Arc/Info to Oracle were the most efficient method measured. Using views had mixed results. If the view contained a SELECT command only with no WHERE clause then the performance was similar to that of a native SQL command. If a WHERE clause was included in the view then this created significant overheads increasing the elapsed time at least fivefold. Elapsed time for queries submitted through the database integrator could not be recorded due to systems failure.

The second part of phase II considered the effect of running different types of transaction in the same environment. To measure the effect of the different transactions two types of query were developed. The first type of query was a simple on-line transaction processing type query, designed to run repeatedly for at least an hour at a time. The second query was a long complex query requiring the use multiple relational joins to retrieve the required data. The two queries were run simultaneously on single nodes in the same environment. This was then repeated on multiple nodes. The effect of low-level transaction processing, running in the background, on the elapsed time for the complex query was negligible, even when both queries were run simultaneously on the same node. A higher level transaction processing (where transaction processing was run on multiple nodes) carried some small level of overhead with it. The main effect was seen when transaction processing occurred on node 0 (the communication node) as well as other nodes on the MCS. This had the effect of increasing the elapse time by approximately 20 percent.

The third part of phase II was the workload and multi-user tests where both the amount of work and the number of users were increased. However, this did not take place due to systems failure.

13.2 Conclusions

From the summary and results above the following conclusions can be drawn.

The performance tests conducted in both phase I and phase II show that a parallel GIS environment consisting of general purpose parallel hardware, a parallel database and commercially available GIS software is a viable solution to the data processing difficulties emerging in corporate GIS. The tests in phase I and phase II have shown that it is not only possible to have each individual element of the environment running successfully, but that it is feasible to have a serial GIS running at the same time as a parallel database, and further that they are able to link up successfully using existing GIS/database interfaces.

Communications are an issue. The performance tests in phase I indicate that there is a real performance overhead when external communications are managed by a single node. The strategy adopted by this research project was to dedicate a single node to the task of handling communications. Another option would be to spread the burden of communication between several nodes. Partitioning the nodes on the MCS was also feasible and caused no perceivable overheads or access problems.

True parallel access has not yet been implemented, and results showed this. This was particularly noticeable when the elapsed time to access data using two nodes was measured. A much more successful strategy was to manipulate the data in the database in a similar fashion to that of a distributed database. This gave much better performance. A truly parallel implementation of the Oracle database has been released with version 8. In this version parallel query processing, query optimisation and indexing have been implemented.

Performance monitoring was made more difficult due to lack of tools available. Although standard UNIX performance monitoring tools and Oracle database monitoring tools were available on the parallel system, they had not been implemented to gather performance statistics for the whole machine. The UNIX tools were limited to providing information about individual nodes. To gain an overall picture it was necessary to develop methods and tools for gathering global performance statistics in the shape of two performance tests harnesses. The harnesses were used to run the tests, measure the performance and gather the data from each node.

The results of the performance tests measuring the influence of the buffer cache on elapsed time for individual queries were unexpected and require further research to understand why some queries took longer to run on the second invocation. The influence of many nodes requesting similar data almost simultaneously was felt to be a factor in the results measured. However, this area of the database requires more research to understand how multiple nodes are using the buffer cache.

The interface between the GIS and the database is perhaps the most important feature in the parallel GIS/database system and a very obvious bottleneck. The performance tests concerning the use of the substantial corporate database demonstrated firstly, that the use of existing interface links between the GIS and the database was feasible and that a corporate hybrid GIS model could form part of an integrated corporate IT strategy. GIS could access data from a data warehouse utilising parallel database system software and, through the Database Integrator, spatial data could be migrated from GIS to the data warehouse. The performance tests demonstrated that the GIS could make use of the parallel facilities offered by MCS and was able to address individual nodes or groups of nodes using the Database Integrator. Finally, the tests demonstrated that data access strategies using views of database tables were successful. However, the issues surrounding the interface between the GIS and the database need to be examined more closely. It was not possible in the research project to explore the Arc/Info – Oracle database interface thoroughly. Native SQL commands issued directly for Arc/Info to Oracle performed the best and a view with a WHERE clause (constructed in Oracle) performed worst. If a corporate parallel GIS/data system were to be a viable option then a closer connection between the GIS and the database needs to be developed to eliminate any inefficiencies in data retrieval and continue to improve performance.

Finally, the tests showed that the parallel environment could support both transaction processing queries and long, complex queries in a single database. The performance overhead measured from running both transaction processing and complex queries in the same environment were minimal. However, it would be necessary to increase both work load and user load to explore the threshold where transaction processing might affect the performance of more complex queries.

The performance tests confirmed that the parallel GIS/database environment is suitable for supporting corporate GIS in an integrated IT strategy with the following recommendations:

- always segregate a node for communication.
- consider implementing the databaset as a distributed system.
- the interface between GIS and database the key to success with this system – needs proper research to vastly improve the performance to make it as efficient as possible.
- performance monitoring tools for parallel systems need to be developed to allow the whole system to be monitored from one application.

13.3 Discussion

The aim of this research project was to investigate alternative approaches to the integration of GIS and corporate database management systems in a parallel environment to identify both potential benefits and risk factors involved in such a strategy. The rapid development of corporate GIS over the last decade had identified many pitfalls to the implementation of GIS, but were only beginning to consider the technological requirements. It was clear from the literature that organisations were keen to make full use of corporate GIS and that there was a rapid expansion of spatial datasets available.

During the rise of corporate GIS, parallel architectures had developed from specialist, scientific machines into more general purpose computers designed with commodity components and utilising operating systems such as SunOS. The move to commodity components was highly significant because it opened the parallel architectures market to many organisations with a need for much greater processing power, but utilising custom built software.

As technology has become more powerful it has been possible to create much large datasets than it is currently feasible for many organisations to process and analyse. The need to process and analyse large datasets (100GB databases are becoming more common) is universal and has encouraged the development of parallel relational database management systems capable of handling very large data volumes. Two GIS approaches, the hybrid and integrated GIS models, suitable for use with parallel relational database technology were identified.

The threads of the research project were beginning to twist together in commercial organisations and it was felt that an investigation into the use of parallel databases for corporate GIS was required.

Accordingly, a test environment was developed consisting of Oracle, a parallel relational database management system, and initially , two GIS – Arc/Info and Smallworld, and later narrowed down to Arc/Info. A two-stage testing strategy was devised to performance test the environment, the first phase to characterise the environment, and the second to test the suitability for corporate GIS use. Two databases were devised for the tests, the first a pilot database constructed from artificial data. The second 'corporate-sized' database to performance tests the environment using real data.

The results showed conclusively that the parallel GIS/database environment would support corporate GIS needs. GIS software designed to run on a single processor could operate alongside parallel database system software and function well in a parallel environment. The GIS was able to link into the parallel database and make use of the substantial data processing power available.

The research also identified a number of areas of further research. The subsequent sections consider those areas and the most recent developments in parallel database technology and the corporate environment that have a bearing on the thesis.

13.4 Corporate GIS/Database Issues

13.4.1 GIS/Database Interface

One of the most important features of this research was the interface between the GIS and parallel database environment. This provides the key to accessing integrated corporate databases, data warehouses, or providing a sustained web-based GIS service. The interface provides the ability to pass SQL commands from the GIS to the database to retrieve selected data. The efficiency of this link impinges not only on the elapsed time the user experiences in making the data request, but also on resource usage, network traffic and disk access. From the user view point the elapsed time is particularly important, especially if GIS functions are hidden behind a user front-end, and they are unaware of the levels of processing being undertaken. From a system viewpoint, inefficient use of resources in an environment dealing with a huge work load, generated by thousands of users, will inevitably lead to severe contention for network facilities and processing resources.

From the performance tests it was clear that the most resource intensive and time consuming part of the transaction was the query execute part. Increased work loads had little effect on either the parse or fetch stages of data retrieval. One solution would be parallel query decomposition, spreading the cost of the execute stage across a number of nodes. Parallel query decomposition is very amenable to long transaction queries where individual queries consume a great deal of CPU and disk I/O (Oracle Corporation 1997a) and would greatly reduce the time taken to execute the query. Transaction processing, however, would not benefit from parallel query decomposition because the overhead involved in breaking the query down would impede throughput. Therefore, in a mixed transaction environment a distinction between the different transactions would have to be made. Oracle, version 8 (1997) fully supports a parallel database server capable of parallel query decomposition.

Using parallel query decomposition to spread the execution load over a number of processors calls into question the performance of the buffer caches, both in the SGA and on the disk controller. From the buffer cache tests in phase I it can be seen that the effect of the two buffer caches together is not only difficult to measure, but also rather uncertain. The usual effect of a buffer is to improve the performance of the second run of a particular query. This is because when the query is run for a second time the data required is already resident in memory and are retrieved faster. However, the results from some of the test runs in phase I were faster on the first run of the query than the second. A network bottleneck may have been the cause for the odd behaviour seen in these tests, but it deserves further investigation.

13.4.2 Transaction Mix

The tests in phase II demonstrated the ability of the environment to support both transaction processing and long, complex queries together, while both accessing the same, significant sized database. The next stage of investigation would require the development of a very large database, in the range of 100GB to consider whether it is feasible to integrate corporate GIS and MIS into such systems. Long transactions performed on tables holding such huge volumes of data may suffer severe performance difficulties.

13.5 Benchmarking Data Warehousing and Web-Based Applications

Since 1998 two benchmarks have been developed for performance testing web-based databases and databases specifically designed for data warehousing roles.

The Transaction Processing Performance Council (TPC) is in the process of producing a transactional web benchmark. The benchmark is designed to represent any business that markets or sells over the internet. It will also benchmark intranet environments which make use of web based transactions for internal operations. The benchmark will concentrate on the performance of systems supporting users browsing, ordering, and conducting transaction oriented business activities (Shanley, 1998). Although the benchmark is aimed at transaction processing type web activities, the processes and decisions that the TPC have evaluated and implemented will prove useful to those developing GIS-web based businesses, or providing access to GIS based applications.

The second benchmark which has been available since April 1995 is TPC-D (TPC, 1995) and is of direct importance for GIS/database benchmarking. TPC-D is aimed at modelling decision support systems in which complex *ad hoc* business-oriented queries are submitted against a large database. The queries might access large portions of the database and typically involve multiple relational joins, extensive sorting, grouping and aggregation and sequential scans. The benchmark assesses the cost/performance of particular systems (TPC, 1998). This represents a huge step forwards in benchmarking for GIS because it measures the typical types of query generated by GIS functions and would be suitable for parallel database analysis.

13.5.1 E-Commerce

The most rapid area of business growth at the moment is e-commerce, the name given to businesses that transact their business over the web. There are a number of obvious examples at the moment, such as air line ticket sales and on-line book and music businesses. Other businesses beginning to flourish are e-law and consultancy services. GIS users, developers and vendors are also beginning to develop on-line GIS services including Spatial Online – a business belonging to ESRI that provides a location for selling on ArcView solutions created by developers, and on-line consultancy e.g. Chesapeake Analytics, providing access to people with all types of GIS/Remote Sensing skills. Other industries with huge GIS potential are travel agencies, tourist information, road traffic information providers such as AA and RAC, job hunting and house hunting agencies. All of these companies have access to a potentially huge market through the internet and will have to sustain user loaded of over 40,000-50,000 users a day. It is imperative that the database technology is capable of doing so.

E-commerce also provides the business with large volumes of data which must have a quick turn around. Much of the data received from the user can be processed and re-used, either to assess how well the company is doing, whether it is appealing to the right audience, and whether their site is easily accessible. However, this information can also be re-packaged and sold on to companies such as those specialising in marketing life style data. It is vital that there is a very quick turn around for both analysis purposes and for profit.

References

Akl, S. G., 1989, *The design and analysis of parallel algorithms*. Prentice Hall International Inc., London.

Alla, P. and Trow, S. W., 1990, Implementation of GIS - a way to optimising utility operations. *AGI'90: GIS - The key to managing information Proceedings*, 2nd National Conference and Exhibition, Brighton, UK.

Amdahl, G., 1967, The validity of the single processor approach to achieving large scale computing capabilities. *AFIPS Conference Spring Joint Computer Conference Proceedings*, **30,** 483-485.

American National Standards Institute (ANSI), 1989, *Database Language SQL*, X3.135-1989.

Anon et al., 1985, A measure of transaction processing power. *Datamation*, **31,** (7), 112-118.

Antennuci, J. C., Brown, K., Croswell, P. L., Kevany, M. J., and Archer, H., 1991, *Geographic Information Systems*. Van Nostrand Reinhold, New York.

Arivav, G., 1986, *A temporally oriented datamodel. ACM Transactions on Database Systems*, **11**, 499-527.

Armstrong, M. P., 1994, GIS and high performance computing. In *Proceedings of GIS/LIS'94*. American Congress on Surveying and Mapping, Bethesda, 4-14.

Armstrong, M. P. and Densham, P. J., 1992, Domain decomposition for parallel processing of spatial problems. *Computers, Environment, and Urban Systems*, **16**, 497-513.

Armstrong, M. P. and Marciano, R. J., 1996, Local interpolation using a distributed parallel super computer. *International Journal of Geographical Information Systems*, **10**, (6), 713-729.

Aronoff, S., 1989, *Geographic Information Systems: a management perspective*. WDL Publications, Ottawa.

Aronson, P., 1985, Applying software engineering to a general purpose Geographic Information System. *Proceedings of AUTOCARTO 7*. ASPRS, Falls Church, Virginia, 346-355.

Astrahan, M., Blasgen, M. W., Chamberlin, D. D., Eswaren, K. P., Gray, N. J., Griffiths, P. P., King, W. F., Lorie, R. A., McJones, P. R., Mehl, J. W., Putzolu, G. R., Traiger, I. L., Wade, B. W. and Watson, V., 1976, System R: relational approach to database management. *ACM Transactions on Database Systems* **1**, (2), 97-137.

Banerjee, J. and Hsiao, D. K., 1978, Concepts and capabilities of a database computer. *ACM Transactions on Database Systems*, **3**, (4), 347-384.

Barr, R., 1997, Hardware heaven? *GIS Europe*, December, 12-13.

Barr, R., 1998, Thin is in. *GIS Europe*, January, 14-15.

Beckett H., 1995, Analysis: parallell computing. *DEC Computing*, 18 October,6-7.

Bitton, D. and Turbyfill, C., 1983, Benchmarking database systems: a systematic approach. *Proc. VLDB Conference 1983*, Florence, Italy.

Branagan, I., 1995, A picture tells a thousand stories. *AGI'95: Expanding your world*, International Conference Centre, Birmingham, UK.

Bridger Systems, 1999, Orbit Database System.
Available from <http://www.bridger-link-usa.com/brhm25.htm>.

Brocklehurst, E. B., (1991), *Survey of benchmarks*. NPL Report DITC 192/91 National Physical Laboratory, Teddington, Middlesex, UK.

Bundock, M., 1987, An integrated DBMS approach to geographic information systems. In Chrisman, N. R. (Ed.), *Auto Carto 8 Proceedings*, Baltimore, MD, 292-301.

Bytes Technology Group, 1997, Novell's NDS for NT. *Talking Technology*, **1,** December. Available from <http://www.thin-client.com/>.

Callaghan, J. G., 1989, GIS - Increasing business efficiency and profit. *AGI'89: GIS - A Corporate Resource, first national conference and exhibition.* National Motorcycle Museum Conference Centre Birmingham, UK, 1.4.1-1.4.4.

Callingham, M., 1995, The structure of catchment areas. *AGI'95: Expanding Your World*, International Conference Centre, Birmingham, UK.

Campbell, H., 1991, *Organisational issues and the utilisation of Geographic Information Systems*. RRL Initiative Discussion Paper 9.

Campbell, H., 1994, How effective are GIS in practice? A case study of British local government. *International Journal of Geographical Information Systems*, **8**, (3), 309-325.

Campbell, H. and Masser, I., 1991, The impact of GIS on Local Government in Great Britain. *AGI'91: third national conference and exhibition*, International Convention Centre, Birmingham, UK, 2.5.1-2.5.6

Campbell, H. and Masser, I., 1992, GIS in local government: some findings from Great Britain. *International Journal of Geographical Information Systems*, **6**, (6), 529-546.

Cattel, R., 1993, An engineering database benchmark. In Gray, J. (Ed.), *The benchmark handbook for database and transaction processing systems.* Morgan Kaufman Publishers, Inc., San Mateo, CA., 2nd edition.

Chan, B. S. B., 1993, *An investigation of distributed database system applications in corporate GIS*. The Department of Geography, The University of Edinburgh, Edinburgh.

Chance, A., Newell, R. G., and Theriault, D. G., 1990, *An overview of MAGIK*, Technical Paper 9/1. SMALLWORLD Systems Ltd, Cambridge, U.K.

Chen, P., 1976, The entity-relationship model - towards a unified view of data. *Association of Computing Machinery, Transactions and Database Systems*, **1**, (1), 9-36.

Clegg, P., 1992, GIS implementation - the Sheffield experience. *AGI'92: World of GIS, fourth national conference and exhibition*, International Convention Centre, Birmingham, UK, 3.9.

Codd, E. F., 1970, A relational model of data for large shared data banks. *Communications of the ACM*, **13**, (6), 377-387.

Corrigan, P. and Gurry, M., 1993, *Oracle Performance Tuning*. O'Reilly, UK.

Curnow, H. J. and Wichmann, B. A., 1976, A synthetic benchmark. *Computer Journal*, **19**, 1, 43-49.

Dangermond, J. and Morehouse S., 1987, Trends in hardware for geographical information systems. *Auto Carto 8 Proceedings*, American Congress for Surveying and Mapping, Bethesda, 380-385.

Date, C. J., 1986, *An introduction to database systems*. 2nd edition, Addison-Wesley, Reading, MA.

Date, C. J., 1988, *An introduction to database systems*. 4th edition Addison-Wesley, Reading, MA.

DebitCredit: a standard? (1986). *FT Systems*, 47, 2-8.

Denness, I., 1997, Data for local authorities. Part I: Discovering what's out there. *Mapping Awareness*. **July**.

Denning, P. J. and Tichy, W. F., 1990, Highly parallel computing. *Science*, **250**, 1217-1222.

Department of the Environment, 1987, *Handling geographic information*. Report of the Committee of Enquiry Chaired by Lord Chorley. Her Majesty's Stationery Office, London.

DeWitt, D. J. and Hawthorn, P. B., 1981, A performance evaluation of database machine architectures. In Zaniolo, C. and Delobel, C. (Eds.), *Proc. 7th Int. Conference on VLDB*, Cannes, France.

DeWitt, D. J., Smith, M. and Boral, H., 1987, *A single-user performance evaluation of the Teradata Database Machine*. MCC Technical Report no. DB-081-87.

DeWitt, D. J., Ghandeharizadeh, S. and Schneider, D., 1988, A performance analysis of the gamma database machine. *Proceedings of the ACM-SIGMOD International Conference on Management of Data*. Chicago.

DeWitt, D. J., *et al.*, 1990, The gamma database machine project. *IEEE Knowledge and Data Engineering*, **2**,1.

DeWitt, D. J. and Gray, J., 1992, Parallel database systems: the future of high performance database systems. *Communications of the ACM*, **35**, 85-98.

DeWitt, D. J., 1993, The Wisconsin Benchmark: past, present, and future. In Gray, J. (Ed.), *The benchmark handbook for database and transaction processing systems.* Morgan Kaufman Publishers, Inc., San Mateo, California.

Dhillon, P., (1992), Towards an integrated future. *AGI'92: World of GIS, fourth national conference and exhibition*, International Convention Centre, Birmingham, UK.

Ding, Y. and Densham, P. J., 1996, Spatial strategies for parallel spatial modelling. *International Journal of Geographical Information Systems*, **10**, (6), 669-698.

Donohoe, P. *et al.*, 1990, *Hartstone benchmark users guide, version 1.0.* Technical Report CMU/SEI-90-UG-1, ESD-90-TR-5, Carnegie Mellon University, March

Dowd, K., 1993, *High performance computing.* O'Reilly & Associates, Inc., Sebastapol, CA.

Dowers, S., Gittings, B.M., Healey, R.G., Sloan, T.M., and Waugh, T.C., 1991, *Characterising GIS performance and the potential for parallelism.* Parallel Architectures Laboratory for GIS, RRL Scotland, Dept. Geography, University of Edinburgh, Edinburgh, Scotland.

Edinburgh Parallel Computing Centre (EPCC), 1992, *Parallel systems: getting the benefits.* Edinburgh.

Egenhofer, M. J., 1991, Extracting SQL for cartographic display. *Cartography and Geographic Information Systems*, **18**, 230-245.

Egenhofer, M. J., 1992, Why not SQL? *International Journal of Geographical Information Systems*, **6**, (2), 71-85.

Englert, S., Gray, J., Kocher, T. and Shah, P., 1989, *A benchmark of NonStop SQL Release 2 demonstrating near-linear speed-up and scaleup on large databases*. Tandem Computers, Technical Report 89.4. Tandem part no. 27469.

ESRI, 1991, *Arc/Info: Managing tabular data*. Environmental Systems Research Institute, Inc., Redlands, CA.

ESRI, 1999, *Spatial Database Engine (SDE) - Product Description.* Available at <http://www.esri.com/software/sde/description.htm>.

Fanner, P., 1997, Tapping into mapping. *Mapping Awareness.* July.

Flynn, M. J., 1972, Some computer organisations and their effectiveness. *IEEE Transactions on Computers*, **C-21**, (9), 948-960.

Fox, G. C., Johnson, M. A., Lyzenga, G. A., Otto, S. W., Salmon, J. K. and Walker, D. W., 1988, *Solving problems on concurrent processors: vol.1, general techniques and regular problems*. Prentice Hall, Englewood Cliffs,NJ.

Gadd, S., 1992, GIS - forging a key role in data integration. *AGI'92: World of GIS, fourth national conference and exhibition*, International Convention Centre, Birmingham, UK.

Ghosh, A., 1999, A new high performance parallel GIS data server. *SUGI Proceedings*. Available at <http://www.sas.com/usergroups/sugi/abstracts/abs112.html>.

Gittings, B. M., Dowers, S., Healey, R. G., Sloan, T. M. and Waugh, T. C., 1991, *Identifying the performance constraints on geographical information systems in the VAXcluster computing environment*. Working Paper 29, Regional Research Laboratory for Scotland, Edinburgh.

Good, B., Homan, P. W., Gawlick, D. E. and Sammer, H., 1985, One thousand transactions per second. *IEEE Compcon Proceedings*, San Francisco.

Goodchild, M. F., 1992, Geographic Information Science. *International Journal of Geographical Information Systems*, **6**, 31-45.

Graefe, G., 1990, Encapsulation of parallelism in the volcano query processing system. *Proceedings of the 1990 ACM-SIGMOD International Conference on Management of Data.*

Gray, J., 1993, *The benchmark handbook for database and transaction processing systems*. Morgan Kaufman Publishers, Inc., San Mateo, CA.

Grimshaw, D. J., 1997, A survey of GIS use in the business sector. *AGI'97: Geographic information - exploiting the benefits*, National Exhibition Centre, Birmingham, UK.

Guptill, S. C., 1986, A new design for the U.S. Geological Survey's National Digital Cartographic Database. In Blakemore, M. (Ed.), *Auto Carto 7 Proceedings*, London, **2,** 10-18.

Guptill, S. C., 1987, Desirable characteristics of a spatial database management system. In Chrisman, N. R. (Ed.), *Auto Carto 8 Proceedings*, Baltimore, MD, 292-301.

Hack, J. J., 1989, On the promise of general purpose parallel computing. *Parallel Computing* **10**, 261-275.

Hanson, R. J., TPC Benchmark B - what does it mean and how to use it. Transaction Processing Council.
Available from <http://www.tpc.org/bdetail.html>

Hartley, J. C., 1990, *Getting started in Digital Records*. Presented to: The Institution of Gas Engineers, North of England Section, May.

Healey, R. G., 1991, Database Management Systems. In Maguire, D. J., Goodchild, M. F., and Rhind, D. W. (Eds.), *Geographical Information Systems: Principles and Applications.* Longman, Harlow.

Healey, R. G., 1996, Special Issue on Parallel Processing in GIS. *International Journal of Geographical Information Systems*, **10**, (6), 667-668,
Available from <http://www.geo.ed.ac.uk>.

Healey, R. G. and Desa, G. B., 1989, Transputer based parallel processing for GIS analysis: problems and potentialities. In *Auto-Carto 9 Proceedings*, American Congress for Surveying and Mapping, Bethesda, 90-99.

Healey, R. G., Dowers, S., Gittings, B. M. and Mineter, M., 1998a, *Parallel Processing Algorithms for GIS.* Taylor & Francis Ltd., London.

Healey, R. G., Dowers, S., Gittings, B. M., Sloan, T. M., *et al.* 1991, Determination of computing resource requirements for GIS processing in a workstation environment. *EGIS 91*, Brussels, Belgium.

Healey, R. G., Dowers, S., Gittings, B. M., and Tranter, M. J., 1998b, Parallel database management systems for GIS. In Healey, R. G., Dowers, S., Gittings, B. M., and Mineter, M. (Eds.), *Parallel Processing Algorithms for GIS.* Taylor & Francis Ltd., London.

Healey, R. G. and Waugh, T. C., 1987, The Geoview design: a relational database approach to geographic data handling. *International Journal of Geographical Information Systems*, **1**, 101-118.

Hey, A. J. G., 1990, The Genesis parallel benchmarks. *In proceedings of first EuroBen workshop*. AFUU.

Heywood, I., 1997, *Beyond Chorley: current geographic information issues*. Association for Geographic Information, London.

Hobson, S. A., 1992, Bridging Islands to increase return on investment. *AGI'92: World of GIS, fourth national conference and exhibition,* International Convention Centre, Birmingham, UK, 1.4.1-1.4.5.

Holman, A. and Barton, 1991, E., *The computing surface - a platform for Oracle*. Technical Paper, A presentation made to the European Oracle User Group, Basel, June.

Homer, I. R., and Watson, M., 1992, Integrated data saves money!! *AGI'92: World of GIS, fourth national conference and exhibition Proceedings*, International Convention Centre, Birmingham, UK, 3.10.1-3.10.5.

Hopkins, S., Healey, R. G. and Waugh, T. C., 1992, Algorithm scalability for line intersection detection in parallel polygon overlay. *In Proceedings of 5th International Symposium on Spatial Data Handling*, International Geographical Union, Columbia, 210-218.

Hornby, R., 1990, Successful application of GIS at Nationwide Anglia. *AGI'90: The key to managing information, 2nd National Conference and Exhibition*, Brighton, UK.

IOBENCH, 1989, Computer Architecture News, August.

Ireland, P., 1997, Plugging into Cyberspace. *GIS Europe*, August.

Ives, M. J., 1985, *Inter-utility exchange of digital records.* Presented to: The Institution of Gas Engineers, South Western Section, Keynsham.

Ives, M. J. 1991, The operation of a digital records system in the United Kingdom. *18th World Gas Conference*, Berlin.

Ives, M. J. 1992, Reaping the benefits. *AGI'92: World of GIS, fourth national conference and exhibition*, International Convention Centre, Birmingham, UK, 1.6.1-1.6.3.

Ives, M. J., 1993, The British Gas Digital Records System. *Mapping Awareness & GIS in Europe,* 7, (3), 25-27.

Jade, 1999.
Available from <http://www.discoverjade.com>.

Jakobek, S., 1990, *Scalable Computing in the 1990s*. Paper presented at the UK Oracle User Group Meeting, 18 September.

Knott, D. R. and Goodall, K., 1991, *Digital Records - theory into practice.* Presented to: The Manchester District Junior Gas Association, Haydock Park.

Lee, S., 1995, Parallel processing in the data warehouse.
Available from
<http://www.developer.ibm.com/library/aixpert/feb95/aixpert_feb95_oracle.html>

Lodwick, A. and Cushnie, J., 1990, A GIS pilot study in Berkshire. *Mapping Awareness*, **4**, December, 39-42.

Lorie, R. and Schek, H. J., 1988, On dynamically defined complex objects and SQL. In Dittrich, K. (Ed.), *Advances in Object-Oriented database systems - Proceedings of the 2nd International Workshop on Object-Oriented Database Systems*, Bad Munster an Stein-Edernberg, Germany, Springer-Verlag, **334,** 323-328, (Lecture Notes in Computer Science).

Maguire, D. J., Goodchild, M. F. and Rhind, D. W., (Eds.), 1991, *Geographical Information Systems: Principles and Applications*. Longman, Harlow.

Mahoney, R. P., 1992, The organisational implications of corporate data management for GIS. *AGI'92: World of GIS, fourth national conference and exhibition*, International Convention Centre, Birmingham, UK**,** 3.5.1-3.5.4.

Mattos, N. M., Meyer-Wegener, K. and Mitschang, B., 1993, Grand tour of concepts for object-orientation from a database point of view. *Data and Knowledge Engineering,* **9**, 321-352.

McLaren, R. A. and Healey, R. G., 1992, Corporate harmony: a review of GIS integration tools. *AGI'92: World of GIS, fourth national conference and exhibition*, International Convention Centre, Birmingham, UK, 1.17.1-1.17.5.

Meiko, 1992, Computing surface: *Oracle on the Unix Performance Mainframe.*

Microsoft Corporation, 1999, *Prologic teams up with Microsoft to deliver a more scalable SQL Server and Windows NT-based banking solution.*
Available at <http://www.microsoft.com/PressPass/comdex/docs/prologic-pr.htm>.

Mierendorff, H. and Trottenbeg, U., 1988, Performance evaluation for Suprenum Systems. In *Evaluating Supercomputers*, Unicom Seminar, 84-97.

Mitchell, M., 1997, Saving for a rainy day. *Mapping Awareness*, July.

Mitschang, B., 1989, Extending the relational algebra to capture complex objects. In Apers, P. and Wiederhold, G. (Eds.), *Fifteenth international conference on very large databases*, Amsterdam, 297-306.

Morehouse, S., 1985, Arc/Info: a geo-relational model for spatial information. *Auto-Carto VII Proceedings*, Washington, DC, 388-397.

Morehouse, S., 1989, The architecture oh Arc/Info. *Proceedings of Auto Carto 9*. ASPRS, Falls Church, Virginia, 266-277.

Morgan, R., 1991, Integrated GIS. In Cadoux-Hudson, J. and Heywood, D. I. (Eds.), *AGI Year Book.* Miles-Arnold, London, 197-200.

Mower, J. E., 1996, Developing parallel procedures for line simplification. *International Journal of Geographical Information Systems*, **10**, (6), 699-712.

Newell, R. G. and Easterfield, M. E., 1990, Version Management - the problem of the long transaction, *Mapping Awareness Conference*, Oxford, January.

Nicol, D. M. and Willard, F. H., 1988, Problem size, parallel architecture and optimal speed-up. *Journal of Parallel and Distributed Computing*, **5**, 404-420.

Nitsche, M. and Reuscher, D., 1997, Second-guessing the saver, *GIS Europe*, November, 34-35.

Noervaag, K., 1999, The vagabond parallel object-oriented database system: versatile support for future applications. March
Available from <http://www.idi.ntnu.no/~noervaag/ptoodb.html>.

Olsen, K. J., and West, J. T., 1999, SAS software and the performance effect of parallel architectures. *SUGI Proceedings*, 290.

Ooi, B., Sacks-Davis, R. and McDonell, K., 1989, Extending a DBMS for geographic applications. *IEEE fifth international conference on data engineering*, Los Angeles, CA, 590-597.

Openshaw, S., Cross, A., Charlton, M., and Brunsdon, C., 1990, Lessons learnt from a post mortem of a failed GIS. *AGI'90: GIS-The Key To Managing Information, 2nd International Conference and Exhibition*, Brighton, UK, 2.3.

Oracle Corporation, 1985a, *Oracle overview and introduction to SQL*. Oracle Systems, Redwood Shore, California.

Oracle Corporation, 1985b, *SQL, the quiet revolution*. Oracle Corporation. Redwood Shore, California.

Oracle Corporation, 1990, *SQL, Language reference manual*. Oracle Systems, Redwood Shore, California.

Oracle Corporation, 1991, *Oracle for the Meiko Computing Surface: installation and users guide (v. 6.2).* . Oracle Corporation. Redwood Shore, California.

Oracle Corporation, 1997a, *Exploiting the geographical component of business using the Oracle 8 spatial cartridge*. An Oracle business white paper. Redwood Shore, California.

Oracle Corporation, 1997b, *Oracle 8 Parallel server: concept and administration*. Oracle Corporation. Redwood Shore, California.

Oracle Corporation, 1997c, *Oracle 8 server SQL reference*. Oracle Systems, Redwood Shore, California.

Oracle Corporation, 1997d, *Oracle 8 spatial cartridge: Advances in relational database technology for spatial data management*. Oracle Technical White Paper Volume 7. Redwood Shore, California.

Ordnance Survey, 1988, *Digital Map Data: Ordnance Survey Transfer Format - incorporating OS1988 Specification.* .

Oxborrow, E., 1986, *Databases and database systems: concepts and issues*. Chartwell-Bratt, Bromley, Kent.

Padding, P., 1991, A GIS infrastructure for the information production process. *EGIS 91 Proceedings*, Brussels, Belgium.

Parsys, 1991, *Oracle parallel server on the Parsys Supernode 1000 series*. Technical Paper, Parsys Ltd, London.

Peel, R., 1997a, Corporate balancing act. *Mapping Awareness*, May.

Peel, R., 1997b, Unifying spatial information. *Mapping Awareness*, July.

Peuquet, D. J., 1988, Representations of geographic space: towards a conceptual synthesis. *Annals of the Association of American Geographers*, **78**, 375-394.

Peuquet, D. J., 1994, Its about time: a conceptual framework for the representation of temporal dynamics in geographic information systems. *Annals of the American Geographers*, **84**, 441-465.

Pirahesh, H., Mohan, C., Cheng, J., Liu, T. S. and Selinger, P., 1990, Parallelism in relational database systems: architectural issues and design approaches. In Lu, H., Ooi, B.-C. and Tan, K.-L. (Eds.), *Query Processing in Parallel Relational Database Systems*. IEEE Computer Society Press, Los Alamitos, CA.

Preston, K. and Uhr, L., 1982, *Multicomputers and image processing: algorithms and programs.* Academic Press, New York.

Quinn, M. J., 1987, *Designing efficient algorithms for parallel computers*. McGraw-Hill, New York.

Quinn, M. J. and Deo, N., 1984, Parallel graph algorithms. *Computing Surveys*, **16**, 319-348.

Raab,F., Kohler, W. and Shah, A., 1998, *Overview of the TPC Benchmark C: The Order-Entry Benchmark.* Transaction Processing Performance Council.
Available at <http://www.tpc.org/cdetail.html>.

Ragsdale, S., 1991, *Parallel Programming*. McGraw-Hill, New York.

Raper, J. and Bundock, M., 1991, UGIX: a layer-based model for a GIS user interface. In Mark, D. and Frank, A. (Eds.), *Cognitive and linguistic aspects of geographic space.* Kluwer Academie, Dor, 449-475.

Raper, J., McCarthy, T. and Williams, N., 1997, Integration of real-time GIS with web-based virtual worlds. *AGI'97: Geographic Information - exploiting the benefits*, National Exhibition Centre, Birmingham, UK, 3.5.1-3.5.3.

Rauen, C., 1993, Mapping the future. *Oracle Magazine*, **vii**, (4), 38-41.

Ricardo, C., 1990, *Database systems: principles, design, & implementation*. Macmillan Publishing Company, New York.

Rideout, T. W., 1992, A practical geographic information system for conservation and rural management. In Rideout, T. W. (Ed.), *Geographic Information Systems and Urban and Rural Planning*. Planning and Environmental Study Group of the Institute of British Geographers.

Roche, S. C. and Gittings, B. M., 1996, Parallel polygon line shading: the quest for more computational power from an existing GIS algorithm. *International Journal of Geographical Information Systems*, **10**, (6), 731-746.

Roberts, J., 1989, The Hertsmere experience. AGI'89 *AGI'89: GIS - A Corporate Resource, first national conference and exhibition.* National Motorcycle Museum Conference Centre Birmingham, UK.

Roussopoulos, N., Faloutos, C. and Sellis, T., 1988, An efficient pictorial database system for PSQL. *IEE Transactions on Software Engineering*, **14**, 630-638.

Rowe, L. A., 1986, A shared object hierarchy. In Stonebraker, M. A. and Rowe, L. A., (Eds.), *The POSTGRES papers*. Memorandum no. UCB/ERL, M86/85. College of Engineering, University of California, Berkeley .

Sahay, S. and Walsham, G., 1996, Implementation of GIS in India: organisational issues and implications. *International Journal for Geographical Information Systems*, **10**, (4), 385-404.

Sarda, N., 1990, Extensions to SQL for historical databases. *IEEE transactions on knowledge and data engineering*, **2**, 220-230.

SAS Institute Inc., 1999, *Scalable performance data server - faster disk access for more timely decision making,* The SAS Institute, Inc., Cary, N.C.
Available at <http://www.sas.com/software/components/spds.htm>.

Sawyer, M., 1998, The development of hardware for parallel processing. Healey, R. G., Dowers, S., Gittings, B. M. and Mineter, M., (Eds.), *Parallel Processing for GIS.* Taylor & Francis Ltd., London.

Sawyer, T., 1993, Doing your own benchmark. In Gray, J. (Ed.), *The benchmark handbook for database and transaction processing systems.* Morgan Kaufman Publishers, Inc., San Mateo, CA.

Scott, C. J., 1989, *Genesis pre-study phase report*. Technical Paper, Southampton University.

Serlin, O., 1993, The history of DebitCredit and the TPC. In Gray, J. (Ed.), *The benchmark handbook for database and transaction processing systems.* Morgan Kaufman Publishers, Inc, San Mateo, CA.

Shanley K, 1998, *History and Overview of the TPC.* Transaction Processing Council. Available at <http://www.tpc.org/articles/tpc.overview.history.1.html>.

Sloan T. M., 1998, Vector-to-Raster Conversion. In Healey, R. G., Dowers, S., Gittings, B. M. and Mineter, M., (Eds.), *Parallel Processing Algorithms for GIS.* Taylor & Francis Ltd, London.

Smallworld Systems Ltd, 1991a, *Magik Documentation*. Burleigh House, 13-15 Newmarket Road, Cambridge.

Smallworld Systems Ltd, 1991b, *GIS User's Guide*. Burleigh House, 13-15 Newmarket Road, Cambridge.

Smallworld Systems Ltd, 1991c, *Customisation Overview Documentation*. Burleigh House, 13-15 Newmarket Road, Cambridge.

Smith, N. S., 1987, Testing of relational databases in Ordnance Survey research and development. *SORSA Symposium*, Durham.

Somerville, 1989, *Software engineering*, 3rd edition. Addison-Wesley, Reading, Massachusetts.

Southampton, 1999, The Genesis benchmark suite.
Available at <http://hpcc.soton.ac.uk/RandD/genesis/genesis.html>.

SPEC, 1998a, *SPEC Frequently Asked Questions.*
Available at <http://www.specbench.org/spec/faq/>.

SPEC, 1998b, *SPEC OSG Frequently Asked Questions.*
Available at <http:www.specbench.org/osg/faq>.

SQL3 Standard Committee, *SQL3 Standard.*

SQL92 Standard Committee, *SQL92 Standard.*

SQL-/MM Standard Committee, *SQL-/MM Standard.*

Statler, S., 1993, How parallel should a database be? *Parallelogram,* **56**, 21-23.

Stonebraker, M., (Ed.), 1994, *Readings in Database Systems*. Morgan Kaufman Publishers, San Mateo, California.

Sybase, 1999, New features of Sybase IQ release 11.2 - parallel IQ backup/restore. Available from <http://www.sybase.com/products/dataware/iq11.html>.

Tomlin, D., 1990, *Geographic Information Systems and Cartographic Modelling*. Prentice Hall, Englewood Cliffs, NJ.

Toon, M., 1997, PC graphics grow up. *Mapping Awareness*, **8**, December.

TPC, 1989, *TPC Benchmark A*. San Jose, CA, Shanley Public Relations.

TPC, 1990, *TPC Benchmark B*. San Jose, CA, Shanley Public Relations.

TPC, 1992, *TPC Benchmark C*. San Jose, CA, Shanley Public Relations.

TPC, 1995, *TPC Benchmark D*. San Jose, CA, Shanley Public Relations.

TPC, 1998, *TPC Benchmark W*. San Jose, CA, Shanley Public Relations.

Trew, A. and Wilson, G., 1991, *Past, present, parallel*. Springer-Verlag, London.

Van der Steen, A., 1991, The benchmark of the EuroBen group. *Parallel Computing*, 1-11.

Van Roessel, J. W. and Fosnight, E.A., 1984, A relational approach to vector data structure conversion. In Marble, D. F. *et. al.* (Eds.), *International Symposium on Spatial Data Handling*, Zurich, **1**, 78-95.

Van Roessel, J. W., 1987, Design of a spatial data structure using the relational normal forms. *International Journal of Geographical Information Systems*, **1**, 33-50.

Webb, G. and Todd, P., 1991, Identifying GIS benefits in the Regional Electricity Companies. *AGI'91:third national conference and exhibition*, International Convention Centre, Birmingham, UK, 1.11.

Weiker, R. P., 1984, Dhrystone: a synthetic systems programming benchmark. *Communications of the ACM*, October, 1013-1030.

Weiker,R. P., 1990, An overview of common benchmarks. *IEEE Computer*, 65-76.

Williams, C., 1999, *The SAS system for OS/390 in a parallel sysplex world,* SAS Institute Inc., Cary, N.C.
Available at <http://www.sas.com/partners/enterprise/ibm/parallel.htm>.

Wilson, D., 1989, Tested metal. *Unix review*, **7**, 1, 97-107.

Wong, Z., 1998, Sears, Roebuck and Co. achieves data warehousing success with parallel SAS application.
Available from <http:www.torrent.com/stories/sears-story.html>.

Worboys, M. F., 1992, A generic model for planar geographic objects. *International Journal of Geographical Information Systems*, **6**, (5), 353-372.

Worboys, M.F., 1994, Object-oriented approaches to geo-referenced information. *International Journal of Geographical Information Systems*, **8**, 45, 385-399.

Xiong, D. and Marble, D. F., 1996, Strategies for real-time spatial analysis using massively parallel SIMD computers: an application to urban traffic flow analysis. *International Journal of Geographical Information Systems*, **10**, (6), 769-789.

Appendices

Appendix A

The following SQL queries are those used in the phase I performance tests. The tests are designed to return fixed numbers of rows. There are two versions of each query, those with '_b' in the title have an index on the buildings table, those with '_c' have an index on the customer table. Only the queries with indexes on the buildings table are shown here. The SQL queries were developed by Ben Chan (1993).

1a.sql

```
select town, category
from b_60000, c_20000
where b_60000.building#=c_20000.building#
and c_20000.building# between 20000 and 40000;
```

a1_1b.sql

```
SELECT  CATEGORY, TOWN
  FROM  B_1000, C_333
 WHERE B_1000.BUILDING# = C_333.BUILDING#
   AND B_1000.BUILDING# BETWEEN 1333 AND 1667;
```

1a10_b.sql

```
SELECT  CATEGORY, TOWN
  FROM  B_10000, C_3333
 WHERE B_10000.BUILDING# = C_3333.BUILDING#
   AND B_10000.BUILDING# BETWEEN 3333 AND 6667;
```

1a15_b.sql

```
SELECT  CATEGORY, TOWN
  FROM  B_15000, C_5000
 WHERE B_15000.BUILDING# = C_5000.BUILDING#
   AND B_15000.BUILDING# BETWEEN 5000 AND 10000;
```

1a2_b.sql

```
SELECT CATEGORY, TOWN
    FROM  B_20000, C_6667
   WHERE B_20000.BUILDING# = C_6667.BUILDING#
     AND B_20000.BUILDING# BETWEEN 6667 AND 13333;
```

1a20_b.sql

```
SELECT  CATEGORY, TOWN
  FROM  B_2000, C_667
 WHERE B_2000.BUILDING# = C_667.BUILDING#
   AND B_2000.BUILDING# BETWEEN 1667 AND 2333;
```

1a25.sql

```
SELECT  CATEGORY, TOWN
  FROM  B_25000, C_8333
 WHERE B_25000.BUILDING# = C_8333.BUILDING#
   AND B_25000.BUILDING# BETWEEN 8333 AND 16666;
```

1a3.sql

```
SELECT  CATEGORY, TOWN
  FROM  B_3000, C_1000
 WHERE B_3000.BUILDING# = C_1000.BUILDING#
   AND B_3000.BUILDING# BETWEEN 1000 AND 2000;
```

1a30_b.sql

```
SELECT  CATEGORY, TOWN
  FROM  B_30000, C_10000
 WHERE B_30000.BUILDING# = C_10000.BUILDING#
   AND B_30000.BUILDING# BETWEEN 10000 AND 20000;
```

1a5_b.sql

```
SELECT  CATEGORY, TOWN
  FROM  B_5000, C_1667
 WHERE B_5000.BUILDING# = C_1667.BUILDING#
   AND B_5000.BUILDING# BETWEEN 1667 AND 3333;
```

1a60_b.sql

```
SELECT  CATEGORY, TOWN
  FROM  B_60000, C_20000
 WHERE B_60000.BUILDING# = C_20000.BUILDING#
   AND B_60000.BUILDING# BETWEEN 20000 AND 40000;
```

The following SQL queries were used in phase I the perform tests requiring a percentage of data to be returned. Again there are two similar queries for each percentage using diffeent indexing strategies. Only one version is shown here.

c60_b15.sql

```
SELECT  CATEGORY, TOWN
  FROM  B_60000, C_20000
 WHERE B_60000.BUILDING# = C_20000.BUILDING#
   AND B_60000.BUILDING# BETWEEN 20000 AND 50000;
```

c60_b20.sql

```
SELECT  CATEGORY, TOWN
  FROM  B_60000, C_20000
 WHERE B_60000.BUILDING# = C_20000.BUILDING#
   AND B_60000.BUILDING# BETWEEN 10000 AND 50000;
```

1c60_b25.sql

```
SELECT  CATEGORY, TOWN
  FROM  B_60000, C_20000
 WHERE B_60000.BUILDING# = C_20000.BUILDING#
   AND B_60000.BUILDING# BETWEEN 5000 AND 55000;
```

1c60_b30.sql

```
SELECT  CATEGORY, TOWN
  FROM  B_60000, C_20000
 WHERE B_60000.BUILDING# = C_20000.BUILDING#
   AND B_60000.BUILDING# BETWEEN 1001 AND 61000;
```

1c60_b5.sql

```
SELECT  CATEGORY, TOWN
  FROM  B_60000, C_20000
 WHERE B_60000.BUILDING# = C_20000.BUILDING#
   AND B_60000.BUILDING# BETWEEN 20000 AND 30000;
```

Appendix B

The following is a list of SQL Code used to create the corporate database:

```
REM
REM This ORACLE V6 RDBMS command file was generated by CASE*Dictionary
REM                            on  18-FEB-94
REM
REM For application system GAS version 1
REM
SET SCAN OFF

REM  Objects being generated in this file are:-
REM TABLE
REM      ACCOUNTS
REM      APPLIANCES
REM      APPLIANCE_MAKES
REM      APPLIANCE_TYPES
REM      BUILDINGS
REM      BUILDING_USES
REM      BUILD_BUILDU
REM      CARRIERS
REM      CARRIER_PIPE
REM      CENSUS
REM      CHARGES
REM      COMPLAINTS
REM      COMPLAINTS_CUSTOMER
REM      COMPLETIONS
REM      COMPONENTS
REM      COMP_APL_M
REM      COMP_APL_T
REM      COUNCIL_TAX
REM      COUNTIES
REM      CUSTOMERS
REM      EMERGENCIES
REM      EMERGENCY_CUSTOMER
REM      EMPLOYEES
REM      GOVERNOR_STATIONS
REM      HOUSEHOLDS
REM      INJECTION_POINTS
REM      INSTALLATIONS
REM      INSTALL_APPL
REM      METERS
REM      METER_TYPES
REM      ORIGINATOR_RECIPIENTS
REM      PAY
REM      PIPES
REM      PIPE_NETWORKS
REM      PIPE_STREET
REM      PLANS
REM      SCHEDULES
REM      SERVICE_APPL
REM      SERVICE_CONTRACTS
REM      SIPHONS
REM      SPECIALIST_CONTRACTORSS
REM      SPECIALIST_SCHEDULE
REM      SPECIAL_CUSTOMER
REM      SPECIAL_NEEDS
REM      STREETS
REM      STREETWORKS
REM      STREETWORK_ORIG
REM      STREETWORK_SCHEDULE
REM      STREET_STREETWORK
REM      TOWNS
REM      USAGE
REM      VEHICLES
REM      VEHICLE_USAGE
REM      VEHICULE_MAKERS

REM
REM      Created from Entity ACCOUNT by OPS$MJT on 18-FEB-94
REM
```

```
PROMPT
PROMPT Creating Table ACCOUNTS
CREATE TABLE accounts(
 acc_id                          NUMBER          NOT NULL,
 acc_customer_id                 NUMBER          NOT NULL,
 acc_aprox_annual_usage          NUMBER          NULL,
 acc_credit_rating               CHAR(20)        NULL,
 acc_end_date                    DATE            NULL,
 acc_start_date                  DATE            NULL,
 acc_payment_due                 NUMBER(9,2)   NULL
)
;

COMMENT ON TABLE accounts
    IS 'Created from Entity ACCOUNT by OPS$MJT on 18-FEB-94';

COMMENT ON COLUMN accounts.acc_id
    IS 'unique id';

COMMENT ON COLUMN accounts.acc_customer_id
    IS 'unique id';

COMMENT ON COLUMN accounts.acc_aprox_annual_usage
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE APROX_ANNUAL_USAGE';

COMMENT ON COLUMN accounts.acc_credit_rating
    IS 'credit worthiness of customer';

COMMENT ON COLUMN accounts.acc_end_date
    IS 'final date of account (end of quarter)';

COMMENT ON COLUMN accounts.acc_start_date
    IS 'date account started (start of quarter)';

COMMENT ON COLUMN accounts.acc_payment_due
    IS 'amount owed by customer';

REM
REM     Created from Entity APPLIANCE by OPS$MJT on 18-FEB-94
REM
PROMPT
PROMPT Creating Table APPLIANCES
CREATE TABLE appliances(
 appl_id                         NUMBER          NOT NULL,
 appl_apl_m_id                   NUMBER          NOT NULL,
 appl_apl_t_id                   NUMBER          NOT NULL,
 appl_customer_id                NUMBER          NULL,
 appl_description                CHAR(40)        NULL,
 appl_make                       CHAR(20)        NULL,
 appl_type                       CHAR(15)        NULL
)
;

COMMENT ON TABLE appliances
    IS 'Created from Entity APPLIANCE by OPS$MJT on 18-FEB-94';

COMMENT ON COLUMN appliances.appl_id
    IS 'unique id for appliance';

COMMENT ON COLUMN appliances.appl_apl_m_id
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE ID';

COMMENT ON COLUMN appliances.appl_apl_t_id
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE ID';

COMMENT ON COLUMN appliances.appl_customer_id
    IS 'unique id';

COMMENT ON COLUMN appliances.appl_description
    IS 'brief description of appliance';

COMMENT ON COLUMN appliances.appl_make
    IS 'name of maker';

COMMENT ON COLUMN appliances.appl_type
    IS 'type of appliance eg. gas fire, cooker etc';
```

```
REM
REM     Created from Entity APPLIANCE MAKE by OPS$MJT on 18-FEB-94
REM
PROMPT
PROMPT Creating Table APPLIANCE_MAKES
CREATE TABLE appliance_makes(
 id                                 NUMBER           NOT NULL,
 name                               CHAR(30)         NULL
)
;

COMMENT ON TABLE appliance_makes
    IS 'Created from Entity APPLIANCE MAKE by OPS$MJT on 18-FEB-94';

COMMENT ON COLUMN appliance_makes.id
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE ID';

COMMENT ON COLUMN appliance_makes.name
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE NAME';

REM
REM     Created from Entity APPLIANCE TYPE by OPS$MJT on 18-FEB-94
REM
PROMPT
PROMPT Creating Table APPLIANCE_TYPES
CREATE TABLE appliance_types(
 apl__id                            NUMBER           NOT NULL,
 apl__name                          CHAR(20)         NULL
)
;

COMMENT ON TABLE appliance_types
    IS 'Created from Entity APPLIANCE TYPE by OPS$MJT on 18-FEB-94';

COMMENT ON COLUMN appliance_types.apl__id
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE ID';

COMMENT ON COLUMN appliance_types.apl__name
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE NAME';

REM
REM     Created from Entity BUILDING by OPS$MJT on 18-FEB-94
REM
PROMPT
PROMPT Creating Table BUILDINGS
CREATE TABLE buildings(
 buil_id                            NUMBER           NOT NULL,
 buil_household_id                  NUMBER           NOT NULL,
 buil_carrier_id                    NUMBER           NULL,
 buil_customer_id                   NUMBER           NULL,
 buil_specialist_id                 NUMBER           NULL,
 buil_street_id                     NUMBER           NULL,
 buil_census_id                     NUMBER           NULL,
 buil_employee_id                   NUMBER           NULL,
 buil_category                      CHAR(15)         NULL,
 buil_label_point                   NUMBER           NULL,
 buil_name                          CHAR(25)         NULL,
 buil_postcode                      CHAR(10)         NULL,
 buil_ycoord                        NUMBER           NULL,
 buil_xcoord                        NUMBER           NULL,
 buil_number                        NUMBER           NULL,
 buil_description                   CHAR(15)         NULL
)
;

COMMENT ON TABLE buildings
    IS 'Created from Entity BUILDING by OPS$MJT on 18-FEB-94';

COMMENT ON COLUMN buildings.buil_id
    IS 'unique id';

COMMENT ON COLUMN buildings.buil_household_id
    IS 'unique id';

COMMENT ON COLUMN buildings.buil_carrier_id
    IS 'unique id';
```

```
COMMENT ON COLUMN buildings.buil_customer_id
    IS 'unique id';

COMMENT ON COLUMN buildings.buil_specialist_id
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE ID';

COMMENT ON COLUMN buildings.buil_street_id
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE ID';

COMMENT ON COLUMN buildings.buil_census_id
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE ID';

COMMENT ON COLUMN buildings.buil_employee_id
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE ID';

COMMENT ON COLUMN buildings.buil_category
    IS 'whether its a listed building or not';

COMMENT ON COLUMN buildings.buil_label_point
    IS 'label point number stored in arc/info';

COMMENT ON COLUMN buildings.buil_name
    IS 'name of building';

COMMENT ON COLUMN buildings.buil_postcode
    IS 'postcode of each individual building';

COMMENT ON COLUMN buildings.buil_ycoord
    IS 'y coordinate position';

COMMENT ON COLUMN buildings.buil_xcoord
    IS 'x coordinate position';

COMMENT ON COLUMN buildings.buil_number
    IS 'Street number of building';

COMMENT ON COLUMN buildings.buil_description
    IS 'description of building eg flat, semi etc';

REM
REM Created by mjt 1/11/94. Table is a temporary table used until I can
REM find time to create data for the rest of the columns
REM

CREATE TABLE temp_build(
 BUIL_ID                                  NUMBER not null
 BUIL_HOUSEHOLD_ID                        NUMBER not null
 BUIL_CARRIER_ID                          NUMBER
 BUIL_CUSTOMER_ID                         NUMBER
 BUIL_SPECIALIST_ID                       NUMBER
 BUIL_STREET_ID                           NUMBER
 BUIL_CENSUS_ID                           NUMBER
 BUIL_EMPLOYEE_ID                         NUMBER
 BUIL_CATEGORY                            CHAR(15)
 BUIL_LABEL_POINT                         NUMBER
 BUIL_NAME                                CHAR(25)
 BUIL_POSTCODE                            CHAR(10)
 BUIL_YCOORD                              NUMBER
 BUIL_XCOORD                              NUMBER
 BUIL_NUMBER                              NUMBER
 BUIL_DESCRIPTION                         CHAR(15)
 COVER                                    NUMBER
)
 pctfree 2
 storage( initial 22728
          next 2280
          pctincrease 0)
;

REM there are no comments for attributes for temp_build, I'm too lazy!

REM
REM     Created from Entity BUILDING USE by OPS$MJT on 18-FEB-94
REM
PROMPT
PROMPT Creating Table BUILDING_USES
CREATE TABLE building_uses(
 id                                 NUMBER           NOT NULL,
 use                                CHAR(20)         NULL
)
;

COMMENT ON TABLE building_uses
    IS 'Created from Entity BUILDING USE by OPS$MJT on 18-FEB-94';

COMMENT ON COLUMN building_uses.id
    IS 'unique id';

COMMENT ON COLUMN building_uses.use
    IS 'description of building use';

REM
REM      AUTOCREATED - Intersection table
REM - changed PCTFREE from 10 to 2 for all occurances in this file
PROMPT
PROMPT Creating Table BUILD_BUILDU
CREATE TABLE build_buildu(
 buil_id1                           NUMBER           NOT NULL,
 id2                                NUMBER           NOT NULL
)
PCTFREE  2
;

COMMENT ON TABLE build_buildu
    IS 'AUTOCREATED - Intersection table';

COMMENT ON COLUMN build_buildu.buil_id1
    IS 'unique id';

COMMENT ON COLUMN build_buildu.id2
    IS 'unique id';

REM
REM      Created from Entity CARRIER by OPS$MJT on 18-FEB-94
REM
PROMPT
PROMPT Creating Table CARRIERS
CREATE TABLE carriers(
 carr_id                            NUMBER           NOT NULL,
 carr_build_id                      NUMBER           NULL,
 carr_contact_name                  CHAR(20)         NULL,
 carr_name                          CHAR(20)         NULL,
 carr_telephone                     CHAR(15)         NULL
)
;

COMMENT ON TABLE carriers
    IS 'Created from Entity CARRIER by OPS$MJT on 18-FEB-94';

COMMENT ON COLUMN carriers.carr_id
    IS 'unique id';

COMMENT ON COLUMN carriers.carr_build_id
    IS 'unique id';

COMMENT ON COLUMN carriers.carr_contact_name
    IS 'name of first point of contact with carrier';

COMMENT ON COLUMN carriers.carr_name
    IS 'name of carrier';

COMMENT ON COLUMN carriers.carr_telephone
    IS 'telephone number';

REM
REM      AUTOCREATED - Intersection table
REM
PROMPT
PROMPT Creating Table CARRIER_PIPE
CREATE TABLE carrier_pipe(
 carr_id1                           NUMBER           NOT NULL,
 pipe_record_id2                    NUMBER(7,0)      NOT NULL
)
PCTFREE  2
```

```
;

COMMENT ON TABLE carrier_pipe
    IS 'AUTOCREATED - Intersection table';

COMMENT ON COLUMN carrier_pipe.carr_id1
    IS 'unique id';

COMMENT ON COLUMN carrier_pipe.pipe_record_id2
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE RECORD_ID';

REM
REM     Created from Entity CENSUS by OPS$MJT on 18-FEB-94
REM
PROMPT
PROMPT Creating Table CENSUS
CREATE TABLE census(
 cens_id                         NUMBER           NOT NULL,
 cens_build_id                   NUMBER           NOT NULL
)
;

COMMENT ON TABLE census
    IS 'Created from Entity CENSUS by OPS$MJT on 18-FEB-94';

COMMENT ON COLUMN census.cens_id
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE ID';

COMMENT ON COLUMN census.cens_build_id
    IS 'unique id';

REM
REM     Created from Entity CHARGE by OPS$MJT on 18-FEB-94
REM
PROMPT
PROMPT Creating Table CHARGES
CREATE TABLE charges(
 char_id                         CHAR(5)          NOT NULL,
 char_rate                       NUMBER           NOT NULL
)
;

COMMENT ON TABLE charges
    IS 'Created from Entity CHARGE by OPS$MJT on 18-FEB-94';

COMMENT ON COLUMN charges.char_id
    IS 'unique id';

COMMENT ON COLUMN charges.char_rate
    IS 'amount band is worth';

REM
REM     Created from Entity COMPLAINTS by OPS$MJT on 18-FEB-94
REM
PROMPT
PROMPT Creating Table COMPLAINTS
CREATE TABLE complaints(
 comp_id                         NUMBER           NOT NULL,
 comp_action                     CHAR(100)        NULL,
 comp_description                CHAR(100)        NULL
)
;

COMMENT ON TABLE complaints
    IS 'Created from Entity COMPLAINTS by OPS$MJT on 18-FEB-94';

COMMENT ON COLUMN complaints.comp_id
    IS 'unique id';

COMMENT ON COLUMN complaints.comp_action
    IS 'description of action taken';

COMMENT ON COLUMN complaints.comp_description
    IS 'description of complaint';

REM
REM     AUTOCREATED - Intersection table
```

```
REM
PROMPT
PROMPT Creating Table COMPLAINTS_CUSTOMER
CREATE TABLE complaints_customer(
 comp_id1                        NUMBER           NOT NULL,
 cust_id2                        NUMBER           NOT NULL
)
PCTFREE  2
;

COMMENT ON TABLE complaints_customer
    IS 'AUTOCREATED - Intersection table';

COMMENT ON COLUMN complaints_customer.comp_id1
    IS 'unique id';

COMMENT ON COLUMN complaints_customer.cust_id2
    IS 'unique id';

REM
REM     Created from Entity COMPLETION by OPS$MJT on 18-FEB-94
REM
PROMPT
PROMPT Creating Table COMPLETIONS
CREATE TABLE completions(
 id                              NUMBER           NOT NULL,
 streetwork_id                   NUMBER           NOT NULL,
 completion_date                 DATE             NULL,
 description                     CHAR(100)        NULL
)
;

COMMENT ON TABLE completions
    IS 'Created from Entity COMPLETION by OPS$MJT on 18-FEB-94';

COMMENT ON COLUMN completions.id
    IS 'unique id';

COMMENT ON COLUMN completions.streetwork_id
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE ID';

COMMENT ON COLUMN completions.completion_date
    IS 'date of completion';

COMMENT ON COLUMN completions.description
    IS 'description of work completed';

REM
REM     Created from Entity COMPONENTS by OPS$MJT on 18-FEB-94
REM
PROMPT
PROMPT Creating Table COMPONENTS
CREATE TABLE components(
 id                              NUMBER           NOT NULL,
 description                     CHAR(50)         NULL
)
;

COMMENT ON TABLE components
    IS 'Created from Entity COMPONENTS by OPS$MJT on 18-FEB-94';

COMMENT ON COLUMN components.id
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE ID';

COMMENT ON COLUMN components.description
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE DESCRIPTION';

REM
REM     AUTOCREATED - Intersection table
REM
PROMPT
PROMPT Creating Table COMP_APL_M
CREATE TABLE comp_apl_m(
 id1                             NUMBER           NOT NULL,
 id2                             NUMBER           NOT NULL
)
PCTFREE  2
```

```
;

COMMENT ON TABLE comp_apl_m
    IS 'AUTOCREATED - Intersection table';

COMMENT ON COLUMN comp_apl_m.id1
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE ID';

COMMENT ON COLUMN comp_apl_m.id2
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE ID';

REM
REM     AUTOCREATED - Intersection table
REM
PROMPT
PROMPT Creating Table COMP_APL_T
CREATE TABLE comp_apl_t(
 id1                              NUMBER           NOT NULL,
 apl__id2                         NUMBER           NOT NULL
)
PCTFREE  2
;

COMMENT ON TABLE comp_apl_t
    IS 'AUTOCREATED - Intersection table';

COMMENT ON COLUMN comp_apl_t.id1
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE ID';

COMMENT ON COLUMN comp_apl_t.apl__id2
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE ID';

REM
REM     Created from Entity COUNCIL TAX by OPS$MJT on 18-FEB-94
REM - changes made by mjt 1/11/94
PROMPT
PROMPT Creating Table COUNCIL_TAX
CREATE TABLE council_tax(
 tax_id                          NUMBER           NOT NULL,
 tax_band                        CHAR(5)          NOT NULL,
 tax_council_id                  NUMBER           NOT NULL,
 tax_household_id                NUMBER           NULL,
 tax_amount_paid                 NUMBER           NULL,
 tax_date                        DATE             NULL
)
 pctfree 2
 storage( initial 34628
          next 3460
          pctincrease 0)
;

COMMENT ON TABLE council_tax
    IS 'Created from Entity COUNCIL TAX by OPS$MJT on 18-FEB-94';

COMMENT ON COLUMN council_tax.tax_id
    IS 'Unique id';

COMMENT ON COLUMN council_tax.tax_band
    IS 'enter letter from A to H';

COMMENT ON COLUMN council_tax.tax_council_id
    IS 'name of council responsible for charging council tax';

COMMENT ON COLUMN council_tax.tax_household_id
    IS 'unique id';

COMMENT ON COLUMN council_tax.tax_amount_paid
    IS 'Amount of money received to date';

COMMENT ON COLUMN council_tax.tax_date
    IS 'date of last payment';

REM
REM     Created from Entity COUNTY by OPS$MJT on 18-FEB-94
REM
PROMPT
PROMPT Creating Table COUNTIES
```

```
CREATE TABLE counties(
 id                                  NUMBER            NOT NULL,
 name                                CHAR(25)          NULL
)
;

COMMENT ON TABLE counties
    IS 'Created from Entity COUNTY by OPS$MJT on 18-FEB-94';

COMMENT ON COLUMN counties.id
    IS 'unique id';

COMMENT ON COLUMN counties.name
    IS 'name of county';

REM
REM     Created from Entity CUSTOMER by OPS$MJT on 18-FEB-94
REM - changes made by mjt 1/11/94
PROMPT
PROMPT Creating Table CUSTOMERS
CREATE TABLE customers(
 cust_id                             NUMBER            NOT NULL,
 cust_surname                        CHAR(50)          NOT NULL,
 cust_title                          CHAR(5)           NOT NULL,
 cust_build_id                       NUMBER            NULL,
 cust_pipe_record_id                 NUMBER(7)            NULL,
 cust_employee_id                    NUMBER            NULL,
 cust_initials                       CHAR(10)          NULL
)
 pctfree 2
 storage(initial 21781
         next 2180
         pctincrease 0)
;

COMMENT ON TABLE customers
    IS 'Created from Entity CUSTOMER by OPS$MJT on 18-FEB-94';

COMMENT ON COLUMN customers.cust_id
    IS 'unique id';

COMMENT ON COLUMN customers.cust_surname
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE SURNAME';

COMMENT ON COLUMN customers.cust_title
    IS 'miss, ms, mr';

COMMENT ON COLUMN customers.cust_build_id
    IS 'unique id';

COMMENT ON COLUMN customers.cust_pipe_record_id
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE RECORD_ID';

COMMENT ON COLUMN customers.cust_employee_id
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE ID';

COMMENT ON COLUMN customers.cust_initials
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE INITIALS';

REM
REM     Created from Entity EMERGENCY by OPS$MJT on 18-FEB-94
REM
PROMPT
PROMPT Creating Table EMERGENCIES
CREATE TABLE emergencies(
 emer_id                             NUMBER            NOT NULL,
 emer_action                         CHAR(100)         NULL,
 emer_description                    CHAR(100)         NULL,
 emer_priority                       NUMBER            NULL
)
;

COMMENT ON TABLE emergencies
    IS 'Created from Entity EMERGENCY by OPS$MJT on 18-FEB-94';

COMMENT ON COLUMN emergencies.emer_id
    IS 'unique id';
```

```
COMMENT ON COLUMN emergencies.emer_action
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE ACTION';

COMMENT ON COLUMN emergencies.emer_description
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE DESCRIPTION';

COMMENT ON COLUMN emergencies.emer_priority
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE PRIORITY';

REM
REM     AUTOCREATED - Intersection table
REM
PROMPT
PROMPT Creating Table EMERGENCY_CUSTOMER
CREATE TABLE emergency_customer(
 emer_id1                        NUMBER              NOT NULL,
 cust_id2                        NUMBER              NOT NULL
)
PCTFREE  2
;

COMMENT ON TABLE emergency_customer
    IS 'AUTOCREATED - Intersection table';

COMMENT ON COLUMN emergency_customer.emer_id1
    IS 'unique id';

COMMENT ON COLUMN emergency_customer.cust_id2
    IS 'unique id';

REM
REM     Created from Entity EMPLOYEE by OPS$MJT on 18-FEB-94
REM - changes made by mjt 1/11/94
PROMPT
PROMPT Creating Table EMPLOYEES
CREATE TABLE employees(
 empl_id                         NUMBER              NOT NULL,
 empl_department                 CHAR(20)            NOT NULL,
 empl_surname                    CHAR(50)            NOT NULL,
 empl_customer_id                NUMBER              NULL,
 empl_pay_id                     NUMBER              NULL,
 empl_schedule_id                NUMBER              NULL,
 empl_initials                   CHAR(10)            NULL,
 empl_job_description            CHAR(100)           NULL,
 empl_status                     CHAR(5)             NULL,
 empl_title                      CHAR(5)             NULL
)
;

COMMENT ON TABLE employees
    IS 'Created from Entity EMPLOYEE by OPS$MJT on 18-FEB-94';

COMMENT ON COLUMN employees.empl_id
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE ID';

COMMENT ON COLUMN employees.empl_department
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE DEPARTMENT';

COMMENT ON COLUMN employees.empl_surname
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE SURNAME';

COMMENT ON COLUMN employees.empl_customer_id
    IS 'unique id';

COMMENT ON COLUMN employees.empl_pay_id
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE ID';

COMMENT ON COLUMN employees.empl_schedule_id
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE ID';

COMMENT ON COLUMN employees.empl_initials
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE INITIALS';

COMMENT ON COLUMN employees.empl_job_description
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE JOB_DESCRIPTION';
```

```
COMMENT ON COLUMN employees.empl_status
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE STATUS';

COMMENT ON COLUMN employees.empl_title
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE TITLE';

REM
REM     Created from Entity GOVERNOR STATION by OPS$MJT on 18-FEB-94
REM
PROMPT
PROMPT Creating Table GOVERNOR_STATIONS
CREATE TABLE governor_stations(
 gove_id                         NUMBER              NOT NULL,
 gove_pipe_record_id             NUMBER(7,0)         NOT NULL,
 gove_date_of_service            DATE                NULL,
 gove_settings                   NUMBER              NULL,
 gove_description                CHAR(100)           NULL
)
;

COMMENT ON TABLE governor_stations
    IS 'Created from Entity GOVERNOR STATION by OPS$MJT on 18-FEB-94';

COMMENT ON COLUMN governor_stations.gove_id
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE ID';

COMMENT ON COLUMN governor_stations.gove_pipe_record_id
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE RECORD_ID';

COMMENT ON COLUMN governor_stations.gove_date_of_service
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE DATE_OF_SERVICE';

COMMENT ON COLUMN governor_stations.gove_settings
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE SETTINGS';

COMMENT ON COLUMN governor_stations.gove_description
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE DESCRIPTION';

REM
REM     Created from Entity HOUSEHOLD by OPS$MJT on 18-FEB-94
REM
PROMPT
PROMPT Creating Table HOUSEHOLDS
CREATE TABLE households(
 hous_id                         NUMBER              NOT NULL,
 hous_surname                    CHAR(20)            NOT NULL,
 hous_title                      char(10),
 hous_initial                    char(10),
 hous_surname                    char(50),
 hous_build_id                   number
)
 pctfree 2
 storage(initial 22438
         next 2250
         pctincrease 0)
;

COMMENT ON TABLE households
    IS 'Created from Entity HOUSEHOLD by OPS$MJT on 18-FEB-94';

COMMENT ON COLUMN households.hous_id
    IS 'unique id';

COMMENT ON COLUMN households.hous_surname
    IS 'family (principle) name of household';

REM table injection_points deleted mjt 1/11/94

REM
REM     Created from Entity INSTALLATION by OPS$MJT on 18-FEB-94
REM
PROMPT
PROMPT Creating Table INSTALLATIONS
CREATE TABLE installations(
 inst_id                         NUMBER              NOT NULL,
 inst_customer_id                NUMBER              NOT NULL,
 inst_cost                       NUMBER(9,2)    NULL,
 inst_start_time                DATE             NULL,
 inst_description               CHAR(100)        NULL,
 inst_end_date                  DATE             NULL,
 inst_start_date                DATE             NULL,
 inst_end_time                  DATE             NULL
)
;

COMMENT ON TABLE installations
    IS 'Created from Entity INSTALLATION by OPS$MJT on 18-FEB-94';

COMMENT ON COLUMN installations.inst_id
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE ID';

COMMENT ON COLUMN installations.inst_customer_id
    IS 'unique id';

COMMENT ON COLUMN installations.inst_cost
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE COST';

COMMENT ON COLUMN installations.inst_start_time
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE START_TIME';

COMMENT ON COLUMN installations.inst_description
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE DESCRIPTION';

COMMENT ON COLUMN installations.inst_end_date
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE END_DATE';

COMMENT ON COLUMN installations.inst_start_date
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE START_DATE';

COMMENT ON COLUMN installations.inst_end_time
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE END_TIME';

REM
REM      AUTOCREATED - Intersection table
REM
PROMPT
PROMPT Creating Table INSTALL_APPL
CREATE TABLE install_appl(
 inst_id1                       NUMBER           NOT NULL,
 appl_id2                       NUMBER           NOT NULL
)
PCTFREE  2
;

COMMENT ON TABLE install_appl
    IS 'AUTOCREATED - Intersection table';

COMMENT ON COLUMN install_appl.inst_id1
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE ID';

COMMENT ON COLUMN install_appl.appl_id2
    IS 'unique id for appliance';

REM
REM      Created from Entity METER by OPS$MJT on 18-FEB-94
REM - changes made by mjt 1/11/94
PROMPT
PROMPT Creating Table METERS
CREATE TABLE meters(
 mete_id                        NUMBER           NOT NULL,
 mete_customer_id               NUMBER           NOT NULL,
 mete_meter_t_id                NUMBER           NOT NULL,
 mete_date                      DATE             NULL,
 mete_reading                   NUMBER           NULL,
)
 pctfree 2
 storage (initial 106395
          next 10640
          pctincrease 0)
;

COMMENT ON TABLE meters
    IS 'Created from Entity METER by OPS$MJT on 18-FEB-94';
```

```
COMMENT ON COLUMN meters.mete_id
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE ID';

COMMENT ON COLUMN meters.mete_customer_id
    IS 'unique id';

COMMENT ON COLUMN meters.mete_meter_t_id
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE ID';

COMMENT ON COLUMN meters.mete_date
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE DATE, contains time of
        reading too';

COMMENT ON COLUMN meters.mete_reading
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE READING';

REM
REM     Created from Entity METER TYPE by OPS$MJT on 18-FEB-94
REM
PROMPT
PROMPT Creating Table METER_TYPES
CREATE TABLE meter_types(
 id                              NUMBER           NOT NULL,
 type                            CHAR(20)         NULL
)
;

COMMENT ON TABLE meter_types
    IS 'Created from Entity METER TYPE by OPS$MJT on 18-FEB-94';

COMMENT ON COLUMN meter_types.id
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE ID';

COMMENT ON COLUMN meter_types.type
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE TYPE';

REM
REM     Created from Entity ORIGINATOR/ RECIPIENT by OPS$MJT on 18-FEB-94
REM
PROMPT
PROMPT Creating Table ORIGINATOR_RECIPIENTS
CREATE TABLE ORIGINATOR_RECIPients(
 orig_id                         NUMBER           NOT NULL,
 orig_name                       CHAR(20)         NULL
)
;

COMMENT ON TABLE ORIGINATOR_RECIPients
    IS 'Created from Entity ORIGINATOR/ RECIPIENT by OPS$MJT on 18-FEB-94';

COMMENT ON COLUMN ORIGINATOR_RECIPients.orig_id
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE ID';

COMMENT ON COLUMN ORIGINATOR_RECIPients.orig_name
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE NAME';

REM
REM     Created from Entity PAY by OPS$MJT on 18-FEB-94
REM
PROMPT
PROMPT Creating Table PAY
CREATE TABLE pay(
 pay_id                          NUMBER           NOT NULL,
 pay_employee_id                 NUMBER           NULL,
 pay_rate                        NUMBER(9,2)    NULL
)
;

COMMENT ON TABLE pay
    IS 'Created from Entity PAY by OPS$MJT on 18-FEB-94';

COMMENT ON COLUMN pay.pay_id
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE ID';

COMMENT ON COLUMN pay.pay_employee_id
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE ID';
```

```
COMMENT ON COLUMN pay.pay_rate
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE RATE';

REM
REM     Created from Entity PIPE by OPS$MJT on 18-FEB-94
REM - changed by mjt 1/11/94
PROMPT
PROMPT Creating Table PIPES
CREATE TABLE pipes(
 pipe_main_id                    NUMBER           not NULL,
 pipe_record_id                  NUMBER           NOT NULL,
 pipe_street_id                  CHAR(8)          NULL,
 pipe_mtr                        CHAR(4)          NULL,
 pipe_material                   CHAR(2)          NULL,
 pipe_length                     NUMBER(8,3)      NULL,
 pipe_joint_type                 CHAR(1)          NULL,
 pipe_dn1                        CHAR(2)          NULL,
 pipe_status                     CHAR(1)          NULL,
 pipe_pressure                   CHAR(2)          NULL,
 pipe_diameter                   NUMBER(6,2)      NULL,
 pipe_district                   CHAR(8)          NULL,
 pipe_diameter_units             CHAR(1)          NULL,
 pipe_action_type                CHAR(1)          NULL
)
;

COMMENT ON TABLE pipes
    IS 'Created from Entity PIPE by OPS$MJT on 18-FEB-94';

COMMENT ON COLUMN pipes.pipe_record_id
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE RECORD_ID';


COMMENT ON COLUMN pipes.pipe_mtr
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE MTR';

COMMENT ON COLUMN pipes.pipe_material
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE MATERIAL';

COMMENT ON COLUMN pipes.pipe_main_id
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE MAIN_ID';

COMMENT ON COLUMN pipes.pipe_length
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE LENGTH';

COMMENT ON COLUMN pipes.pipe_joint_type
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE JOINT_TYPE';

COMMENT ON COLUMN pipes.pipe_dn1
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE DN1';

COMMENT ON COLUMN pipes.pipe_status
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE STATUS';

COMMENT ON COLUMN pipes.pipe_pressure
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE PRESSURE';

COMMENT ON COLUMN pipes.pipe_diameter
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE DIAMETER';

COMMENT ON COLUMN pipes.pipe_district
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE DISTRICT';

COMMENT ON COLUMN pipes.pipe_diameter_units
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE DIAMETER_UNITS';

COMMENT ON COLUMN pipes.pipe_action_type
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE ACTION_TYPE';

REM
REM     Created from Entity PIPE NETWORK by OPS$MJT on 18-FEB-94
REM - changed by mjt 1/11/94
PROMPT
PROMPT Creating Table PIPE_NETWORKS
CREATE TABLE pipe_networks(
 network_id                              NUMBER            NOT NULL,
 tnode                                   NUMBER          NULL,
 fnode                                    NUMBER          NULL,
 length                                   NUMBER(8,3)     NULL,
 pipe_record_id                           NUMBER          NULL
 pipe_network                             number,
)
;

COMMENT ON TABLE pipe_networks
    IS 'Created from Entity PIPE NETWORK by OPS$MJT on 18-FEB-94';

COMMENT ON COLUMN pipe_networks.id
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE ID';

COMMENT ON COLUMN pipe_networks.connectivity_details
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE CONNECTIVITY_DETAILS';

COMMENT ON COLUMN pipe_networks.name
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE NAME';

COMMENT ON COLUMN pipe_networks.total_length
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE TOTAL_LENGTH';

REM
REM     AUTOCREATED - Intersection table
REM
PROMPT
PROMPT Creating Table PIPE_STREET
CREATE TABLE pipe_street(
 pipe_record_id1                  NUMBER(7,0)       NOT NULL,
 stre_id2                         NUMBER            NOT NULL
)
PCTFREE  2
;

COMMENT ON TABLE pipe_street
    IS 'AUTOCREATED - Intersection table';

COMMENT ON COLUMN pipe_street.pipe_record_id1
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE RECORD_ID';

COMMENT ON COLUMN pipe_street.stre_id2
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE ID';

REM
REM     Created from Entity PLANS by OPS$MJT on 18-FEB-94
REM
PROMPT
PROMPT Creating Table PLANS
CREATE TABLE plans(
 plan_id                          NUMBER            NOT NULL,
 plan_streetwork_id               NUMBER            NOT NULL,
 plan_blue_print                  LONG RAW          NULL,
 plan_description                 CHAR(100)         NULL
)
;

COMMENT ON TABLE plans
    IS 'Created from Entity PLANS by OPS$MJT on 18-FEB-94';

COMMENT ON COLUMN plans.plan_id
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE ID';

COMMENT ON COLUMN plans.plan_streetwork_id
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE ID';

COMMENT ON COLUMN plans.plan_blue_print
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE BLUE_PRINT';

COMMENT ON COLUMN plans.plan_description
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE DESCRIPTION';

REM
REM     Created from Entity SCHEDULE by OPS$MJT on 18-FEB-94
REM
PROMPT
PROMPT Creating Table SCHEDULES
CREATE TABLE schedules(
 sche_id                        NUMBER           NOT NULL,
 sche_employee_id               NUMBER           NULL
)
;

COMMENT ON TABLE schedules
    IS 'Created from Entity SCHEDULE by OPS$MJT on 18-FEB-94';

COMMENT ON COLUMN schedules.sche_id
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE ID';

COMMENT ON COLUMN schedules.sche_employee_id
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE ID';

REM
REM     AUTOCREATED - Intersection table
REM
PROMPT
PROMPT Creating Table SERVICE_APPL
CREATE TABLE service_appl(
 serv_id1                       NUMBER           NOT NULL,
 appl_id2                       NUMBER           NOT NULL
)
PCTFREE  2
;

COMMENT ON TABLE service_appl
    IS 'AUTOCREATED - Intersection table';

COMMENT ON COLUMN service_appl.serv_id1
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE ID';

COMMENT ON COLUMN service_appl.appl_id2
    IS 'unique id for appliance';

REM
REM     Created from Entity SERVICE CONTRACTS by OPS$MJT on 18-FEB-94
REM
PROMPT
PROMPT Creating Table SERVICE_CONTRACTS
CREATE TABLE service_contracts(
 serv_id                        NUMBER           NOT NULL,
 serv_customer_id               NUMBER           NOT NULL,
 serv_cost                      NUMBER(9,2)   NULL,
 serv_description               CHAR(100)        NULL,
 serv_end_date                  DATE             NULL,
 serv_start_time                DATE             NULL,
 serv_start_date                DATE             NULL,
 serv_end_time                  DATE             NULL
)
;

COMMENT ON TABLE service_contracts
    IS 'Created from Entity SERVICE CONTRACTS by OPS$MJT on 18-FEB-94';

COMMENT ON COLUMN service_contracts.serv_id
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE ID';

COMMENT ON COLUMN service_contracts.serv_customer_id
    IS 'unique id';

COMMENT ON COLUMN service_contracts.serv_cost
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE COST';

COMMENT ON COLUMN service_contracts.serv_description
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE DESCRIPTION';

COMMENT ON COLUMN service_contracts.serv_end_date
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE END_DATE';

COMMENT ON COLUMN service_contracts.serv_start_time
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE START_TIME';

COMMENT ON COLUMN service_contracts.serv_start_date
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE START_DATE';

COMMENT ON COLUMN service_contracts.serv_end_time
```

```
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE END_TIME';

REM
REM     Created from Entity SIPHON by OPS$MJT on 18-FEB-94
REM
PROMPT
PROMPT Creating Table SIPHONS
CREATE TABLE siphons(
 siph_id                         NUMBER           NOT NULL,
 siph_pipe_record_id             NUMBER(7,0)      NOT NULL,
 siph_date                       DATE             NULL,
 siph_photo                      LONG RAW         NULL,
 siph_settings                   CHAR(100)        NULL,
 siph_description                CHAR(100)        NULL
)
;

COMMENT ON TABLE siphons
    IS 'Created from Entity SIPHON by OPS$MJT on 18-FEB-94';

COMMENT ON COLUMN siphons.siph_id
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE ID';

COMMENT ON COLUMN siphons.siph_pipe_record_id
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE RECORD_ID';

COMMENT ON COLUMN siphons.siph_date
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE DATE';

COMMENT ON COLUMN siphons.siph_photo
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE PHOTO';

COMMENT ON COLUMN siphons.siph_settings
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE SETTINGS';

COMMENT ON COLUMN siphons.siph_description
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE DESCRIPTION';

REM
REM     Created from Entity SPECIALIST CONTRACTORS by OPS$MJT on 18-FEB-94
REM
PROMPT
PROMPT Creating Table SPECIALIST_CONTRACTORSS
CREATE TABLE specialist_contractorss(
 spec_id                         NUMBER           NOT NULL,
 spec_build_id                   NUMBER           NULL,
 spec_contact_name               CHAR(20)         NULL,
 spec_name                       CHAR(30)         NULL,
 spec_telephone                  CHAR(15)         NULL
)
;

COMMENT ON TABLE specialist_contractorss
    IS 'Created from Entity SPECIALIST CONTRACTORS by OPS$MJT on 18-FEB-94';

COMMENT ON COLUMN specialist_contractorss.spec_id
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE ID';

COMMENT ON COLUMN specialist_contractorss.spec_build_id
    IS 'unique id';

COMMENT ON COLUMN specialist_contractorss.spec_contact_name
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE CONTACT_NAME';

COMMENT ON COLUMN specialist_contractorss.spec_name
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE NAME';

COMMENT ON COLUMN specialist_contractorss.spec_telephone
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE TELEPHONE';

REM
REM     AUTOCREATED - Intersection table
REM
PROMPT
PROMPT Creating Table SPECIALIST_SCHEDULE
CREATE TABLE specialist_schedule(
 spec_id1                        NUMBER           NOT NULL,
 sche_id2                           NUMBER           NOT NULL
)
PCTFREE  2
;

COMMENT ON TABLE specialist_schedule
    IS 'AUTOCREATED - Intersection table';

COMMENT ON COLUMN specialist_schedule.spec_id1
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE ID';

COMMENT ON COLUMN specialist_schedule.sche_id2
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE ID';

REM
REM      AUTOCREATED - Intersection table
REM
PROMPT
PROMPT Creating Table SPECIAL_CUSTOMER
CREATE TABLE special_customer(
 id1                                NUMBER           NOT NULL,
 cust_id2                           NUMBER           NOT NULL
)
PCTFREE  2
;

COMMENT ON TABLE special_customer
    IS 'AUTOCREATED - Intersection table';

COMMENT ON COLUMN special_customer.id1
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE ID';

COMMENT ON COLUMN special_customer.cust_id2
    IS 'unique id';

REM
REM      Created from Entity SPECIAL NEEDS by OPS$MJT on 18-FEB-94
REM
PROMPT
PROMPT Creating Table SPECIAL_NEEDS
CREATE TABLE special_needs(
 id                                 NUMBER           NOT NULL,
 priority                           NUMBER           NULL,
 type                               CHAR(30)         NULL
)
;

COMMENT ON TABLE special_needs
    IS 'Created from Entity SPECIAL NEEDS by OPS$MJT on 18-FEB-94';

COMMENT ON COLUMN special_needs.id
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE ID';

COMMENT ON COLUMN special_needs.priority
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE PRIORITY';

COMMENT ON COLUMN special_needs.type
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE TYPE';

REM
REM      Created from Entity STREET by OPS$MJT on 18-FEB-94
REM
PROMPT
PROMPT Creating Table STREETS
CREATE TABLE streets(
 stre_id                            NUMBER           NOT NULL,
 stre_town_id                       NUMBER           NULL,
 stre_name                          CHAR(30)         NULL,
 stre_route_number                  CHAR(5)          NULL,
 stre_type                          CHAR(10)         NULL
)
;

COMMENT ON TABLE streets
    IS 'Created from Entity STREET by OPS$MJT on 18-FEB-94';

COMMENT ON COLUMN streets.stre_id
```

```
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE ID';

COMMENT ON COLUMN streets.stre_town_id
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE ID';

COMMENT ON COLUMN streets.stre_name
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE NAME';

COMMENT ON COLUMN streets.stre_route_number
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE ROUTE_NUMBER';

COMMENT ON COLUMN streets.stre_type
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE TYPE';

REM
REM     Created from Entity STREETWORKS by OPS$MJT on 18-FEB-94
REM
PROMPT
PROMPT Creating Table STREETWORKS
CREATE TABLE streetworks(
 id                                NUMBER           NOT NULL,
 comments                          LONG             NULL,
 notice_period                     CHAR(20)         NULL,
 recip_cross_ref                   NUMBER           NULL,
 start_date                        DATE             NULL,
 start_time                        DATE             NULL,
 reinstatement                     CHAR(20)         NULL,
 orig_cross_ref                    NUMBER           NULL,
 description                       CHAR(100)        NULL,
 end_date                          DATE             NULL,
 end_time                          DATE             NULL
)
;

COMMENT ON TABLE streetworks
    IS 'Created from Entity STREETWORKS by OPS$MJT on 18-FEB-94';

COMMENT ON COLUMN streetworks.id
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE ID';

COMMENT ON COLUMN streetworks.comments
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE COMMENTS';

COMMENT ON COLUMN streetworks.notice_period
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE NOTICE_PERIOD';

COMMENT ON COLUMN streetworks.recip_cross_ref
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE RECIP_CROSS_REF';

COMMENT ON COLUMN streetworks.start_date
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE START_DATE';

COMMENT ON COLUMN streetworks.start_time
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE START_TIME';

COMMENT ON COLUMN streetworks.reinstatement
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE REINSTATEMENT';

COMMENT ON COLUMN streetworks.orig_cross_ref
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE ORIG_CROSS_REF';

COMMENT ON COLUMN streetworks.description
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE DESCRIPTION';

COMMENT ON COLUMN streetworks.end_date
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE END_DATE';

COMMENT ON COLUMN streetworks.end_time
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE END_TIME';

REM
REM     AUTOCREATED - Intersection table
REM
PROMPT
PROMPT Creating Table STREETWORK_ORIG
CREATE TABLE streetwork_orig(
 id1                               NUMBER           NOT NULL,
 orig_id2                         NUMBER           NOT NULL
)
PCTFREE  2
;

COMMENT ON TABLE streetwork_orig
    IS 'AUTOCREATED - Intersection table';

COMMENT ON COLUMN streetwork_orig.id1
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE ID';

COMMENT ON COLUMN streetwork_orig.orig_id2
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE ID';

REM
REM     AUTOCREATED - Intersection table
REM
PROMPT
PROMPT Creating Table STREETWORK_SCHEDULE
CREATE TABLE streetwork_schedule(
 id1                              NUMBER           NOT NULL,
 sche_id2                         NUMBER           NOT NULL
)
PCTFREE  2
;

COMMENT ON TABLE streetwork_schedule
    IS 'AUTOCREATED - Intersection table';

COMMENT ON COLUMN streetwork_schedule.id1
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE ID';

COMMENT ON COLUMN streetwork_schedule.sche_id2
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE ID';

REM
REM     AUTOCREATED - Intersection table
REM
PROMPT
PROMPT Creating Table STREET_STREETWORK
CREATE TABLE street_streetwork(
 stre_id1                         NUMBER           NOT NULL,
 id2                              NUMBER           NOT NULL
)
PCTFREE  2
;

COMMENT ON TABLE street_streetwork
    IS 'AUTOCREATED - Intersection table';

COMMENT ON COLUMN street_streetwork.stre_id1
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE ID';

COMMENT ON COLUMN street_streetwork.id2
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE ID';

REM
REM     Created from Entity TOWN by OPS$MJT on 18-FEB-94
REM
PROMPT
PROMPT Creating Table TOWNS
CREATE TABLE towns(
 town_id                          NUMBER           NOT NULL,
 town_county_id                   NUMBER           NOT NULL,
 town_name                        CHAR(50)         NULL
)
;

COMMENT ON TABLE towns
    IS 'Created from Entity TOWN by OPS$MJT on 18-FEB-94';

COMMENT ON COLUMN towns.town_id
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE ID';

COMMENT ON COLUMN towns.town_county_id
    IS 'unique id';
```

```
COMMENT ON COLUMN towns.town_name
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE NAME';

REM
REM     Created from Entity USAGE by OPS$MJT on 18-FEB-94
REM
PROMPT
PROMPT Creating Table USAGE
CREATE TABLE usage(
 usag_id                         NUMBER           NOT NULL,
 usag_description                CHAR(50)         NULL
)
;

COMMENT ON TABLE usage
    IS 'Created from Entity USAGE by OPS$MJT on 18-FEB-94';

COMMENT ON COLUMN usage.usag_id
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE ID';

COMMENT ON COLUMN usage.usag_description
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE DESCRIPTION';

REM
REM     Created from Entity VEHICLE by OPS$MJT on 18-FEB-94
REM
PROMPT
PROMPT Creating Table VEHICLES
CREATE TABLE vehicles(
 vehi_id                         NUMBER           NOT NULL,
 vehi_schedule_id                NUMBER           NOT NULL,
 vehi_v_make_id                  NUMBER           NOT NULL,
 vehi_comments                   CHAR(100)        NULL,
 vehi_description                CHAR(100)        NULL,
 vehi_number_plate               CHAR(15)         NULL
)
;

COMMENT ON TABLE vehicles
    IS 'Created from Entity VEHICLE by OPS$MJT on 18-FEB-94';

COMMENT ON COLUMN vehicles.vehi_id
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE ID';

COMMENT ON COLUMN vehicles.vehi_schedule_id
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE ID';

COMMENT ON COLUMN vehicles.vehi_v_make_id
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE ID';

COMMENT ON COLUMN vehicles.vehi_comments
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE COMMENTS';

COMMENT ON COLUMN vehicles.vehi_description
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE DESCRIPTION';

COMMENT ON COLUMN vehicles.vehi_number_plate
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE NUMBER_PLATE';

REM
REM     AUTOCREATED - Intersection table
REM
PROMPT
PROMPT Creating Table VEHICLE_USAGE
CREATE TABLE vehicle_usage(
 vehi_id1                        NUMBER           NOT NULL,
 usag_id2                        NUMBER           NOT NULL
)
PCTFREE  2
;

COMMENT ON TABLE vehicle_usage
    IS 'AUTOCREATED - Intersection table';

COMMENT ON COLUMN vehicle_usage.vehi_id1
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE ID';
```

```
COMMENT ON COLUMN vehicle_usage.usag_id2
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE ID';

REM
REM     Created from Entity VEHICULE MAKER by OPS$MJT on 18-FEB-94
REM
PROMPT
PROMPT Creating Table VEHICULE_MAKERS
CREATE TABLE vehicule_makers(
 v_ma_id                          NUMBER           NOT NULL,
 v_ma_name                        CHAR(20)         NULL
)
;

COMMENT ON TABLE vehicule_makers
    IS 'Created from Entity VEHICULE MAKER by OPS$MJT on 18-FEB-94';

COMMENT ON COLUMN vehicule_makers.v_ma_id
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE ID';

COMMENT ON COLUMN vehicule_makers.v_ma_name
    IS 'COLUMN DEFINITION DERIVED FROM ATTRIBUTE NAME';

PROMPT Adding PRIMARY Constraint To ACCOUNTS Table

ALTER TABLE ACCOUNTS ADD (
      PRIMARY KEY (ACC_ID)
      CONSTRAINT ACC_PK
)
/

PROMPT Adding PRIMARY Constraint To APPLIANCES Table

ALTER TABLE APPLIANCES ADD (
      PRIMARY KEY (APPL_ID)
      CONSTRAINT APPL_PK
)
/

PROMPT Adding PRIMARY Constraint To APPLIANCE_MAKES Table

ALTER TABLE APPLIANCE_MAKES ADD (
      PRIMARY KEY (ID)
      CONSTRAINT APL_M_PK
)
/

PROMPT Adding PRIMARY Constraint To APPLIANCE_TYPES Table

ALTER TABLE APPLIANCE_TYPES ADD (
      PRIMARY KEY (APL__ID)
      CONSTRAINT APL_T_PK
)
/

PROMPT Adding PRIMARY Constraint To BUILDINGS Table

ALTER TABLE BUILDINGS ADD (
      PRIMARY KEY (BUIL_ID)
      CONSTRAINT BUILD_PK
)
/

PROMPT Adding PRIMARY Constraint To BUILDING_USES Table

ALTER TABLE BUILDING_USES ADD (
      PRIMARY KEY (ID)
      CONSTRAINT BUILDU_PK
)
/

PROMPT Adding PRIMARY Constraint To BUILD_BUILDU Table

ALTER TABLE BUILD_BUILDU ADD (
      PRIMARY KEY (BUIL_ID1,
                   ID2)
      CONSTRAINT BUILD_BUILDU_PK
)
/

PROMPT Adding PRIMARY Constraint To CARRIERS Table

ALTER TABLE CARRIERS ADD (
      PRIMARY KEY (CARR_ID)
      CONSTRAINT CARRIER_PK
)
/

PROMPT Adding PRIMARY Constraint To CARRIER_PIPE Table

ALTER TABLE CARRIER_PIPE ADD (
      PRIMARY KEY (CARR_ID1,
                   PIPE_RECORD_ID2)
      CONSTRAINT CARRIER_PIPE_PK
)
/

PROMPT Adding PRIMARY Constraint To CENSUS Table

ALTER TABLE CENSUS ADD (
      PRIMARY KEY (CENS_ID)
      CONSTRAINT CENSUS_PK
)
/

PROMPT Adding PRIMARY Constraint To CHARGES Table

ALTER TABLE CHARGES ADD (
      PRIMARY KEY (CHAR_ID)
      CONSTRAINT CHARGE_PK
)
/

PROMPT Adding PRIMARY Constraint To COMPLAINTS Table

ALTER TABLE COMPLAINTS ADD (
      PRIMARY KEY (COMP_ID)
      CONSTRAINT COMPLAINTS_PK
)
/

PROMPT Adding PRIMARY Constraint To COMPLAINTS_CUSTOMER Table

ALTER TABLE COMPLAINTS_CUSTOMER ADD (
      PRIMARY KEY (COMP_ID1,
                   CUST_ID2)
      CONSTRAINT COMPLAINTS_CUSTOMER_PK
)
/

PROMPT Adding PRIMARY Constraint To COMPLETIONS Table

ALTER TABLE COMPLETIONS ADD (
      PRIMARY KEY (ID)
      CONSTRAINT COMPLETION_PK
)
/

PROMPT Adding PRIMARY Constraint To COMPONENTS Table

ALTER TABLE COMPONENTS ADD (
      PRIMARY KEY (ID)
      CONSTRAINT COMP_PK
)
/

PROMPT Adding PRIMARY Constraint To COMP_APL_M Table

ALTER TABLE COMP_APL_M ADD (
      PRIMARY KEY (ID1,
                   ID2)
      CONSTRAINT COMP_APL_M_PK
)
/
```

```
PROMPT Adding PRIMARY Constraint To COMP_APL_T Table

ALTER TABLE COMP_APL_T ADD (
      PRIMARY KEY (ID1,
                   APL__ID2)
      CONSTRAINT COMP_APL_T_PK
)
/

PROMPT Adding PRIMARY Constraint To COUNCIL_TAX Table

ALTER TABLE COUNCIL_TAX ADD (
      PRIMARY KEY (COUN_ID)
      CONSTRAINT COUNCIL_PK
)
/

PROMPT Adding PRIMARY Constraint To COUNTIES Table

ALTER TABLE COUNTIES ADD (
      PRIMARY KEY (ID)
      CONSTRAINT COUNTY_PK
)
/

PROMPT Adding PRIMARY Constraint To CUSTOMERS Table

ALTER TABLE CUSTOMERS ADD (
      PRIMARY KEY (CUST_ID)
      CONSTRAINT CUSTOMER_PK
)
/

PROMPT Adding PRIMARY Constraint To EMERGENCIES Table

ALTER TABLE EMERGENCIES ADD (
      PRIMARY KEY (EMER_ID)
      CONSTRAINT EMERGENCY_PK
)
/

PROMPT Adding PRIMARY Constraint To EMERGENCY_CUSTOMER Table

ALTER TABLE EMERGENCY_CUSTOMER ADD (
      PRIMARY KEY (EMER_ID1,
                   CUST_ID2)
      CONSTRAINT EMERGENCY_CUSTOMER_PK
)
/

PROMPT Adding PRIMARY Constraint To EMPLOYEES Table

ALTER TABLE EMPLOYEES ADD (
      PRIMARY KEY (EMPL_ID)
      CONSTRAINT EMPLOYEE_PK
)
/

PROMPT Adding PRIMARY Constraint To GOVERNOR_STATIONS Table

ALTER TABLE GOVERNOR_STATIONS ADD (
      PRIMARY KEY (GOVE_ID)
      CONSTRAINT GOVERNOR_PK
)
/

PROMPT Adding PRIMARY Constraint To HOUSEHOLDS Table

ALTER TABLE HOUSEHOLDS ADD (
      PRIMARY KEY (HOUS_ID)
      CONSTRAINT HOUSEHOLD_PK
)
/

PROMPT Adding PRIMARY Constraint To INJECTION_POINTS Table
```

```
ALTER TABLE INJECTION_POINTS ADD (
      PRIMARY KEY (INJE_ID)
      CONSTRAINT INJECTION_PK
)
/

PROMPT Adding PRIMARY Constraint To INSTALLATIONS Table

ALTER TABLE INSTALLATIONS ADD (
      PRIMARY KEY (INST_ID)
      CONSTRAINT INSTALLATI_PK
)
/

PROMPT Adding PRIMARY Constraint To INSTALL_APPL Table

ALTER TABLE INSTALL_APPL ADD (
      PRIMARY KEY (INST_ID1,
                   APPL_ID2)
      CONSTRAINT INSTALL_APPL_PK
)
/

PROMPT Adding PRIMARY Constraint To METERS Table

ALTER TABLE METERS ADD (
      PRIMARY KEY (METE_ID)
      CONSTRAINT METER_PK
)
/

PROMPT Adding PRIMARY Constraint To METER_TYPES Table

ALTER TABLE METER_TYPES ADD (
      PRIMARY KEY (ID)
      CONSTRAINT METER_T_PK
)
/

PROMPT Adding PRIMARY Constraint To ORIGINATOR_RECIPIENTS Table

ALTER TABLE ORIGINATOR_RECIPIENTS ADD (
      PRIMARY KEY (ORIG_ID)
      CONSTRAINT ORIG_PK
)
/

PROMPT Adding PRIMARY Constraint To PAY Table

ALTER TABLE PAY ADD (
      PRIMARY KEY (PAY_ID)
      CONSTRAINT PAY_PK
)
/

PROMPT Adding PRIMARY Constraint To PIPES Table

ALTER TABLE PIPES ADD (
      PRIMARY KEY (PIPE_RECORD_ID)
      CONSTRAINT PIPE_PK
)
/

PROMPT Adding PRIMARY Constraint To PIPE_NETWORKS Table

ALTER TABLE PIPE_NETWORKS ADD (
      PRIMARY KEY (ID)
      CONSTRAINT PIPE_NETWO_PK
)
/

PROMPT Adding PRIMARY Constraint To PIPE_STREET Table

ALTER TABLE PIPE_STREET ADD (
      PRIMARY KEY (PIPE_RECORD_ID1,
                   STRE_ID2)
      CONSTRAINT PIPE_STREET_PK
)
/

PROMPT Adding PRIMARY Constraint To PLANS Table

ALTER TABLE PLANS ADD (
      PRIMARY KEY (PLAN_ID)
      CONSTRAINT PLANS_PK
)
/

PROMPT Adding PRIMARY Constraint To SCHEDULES Table

ALTER TABLE SCHEDULES ADD (
      PRIMARY KEY (SCHE_ID)
      CONSTRAINT SCHEDULE_PK
)
/

PROMPT Adding PRIMARY Constraint To SERVICE_APPL Table

ALTER TABLE SERVICE_APPL ADD (
      PRIMARY KEY (SERV_ID1,
                   APPL_ID2)
      CONSTRAINT SERVICE_APPL_PK
)
/

PROMPT Adding PRIMARY Constraint To SERVICE_CONTRACTS Table

ALTER TABLE SERVICE_CONTRACTS ADD (
      PRIMARY KEY (SERV_ID)
      CONSTRAINT SERVICE_CO_PK
)
/

PROMPT Adding PRIMARY Constraint To SIPHONS Table

ALTER TABLE SIPHONS ADD (
      PRIMARY KEY (SIPH_ID)
      CONSTRAINT SIPHON_PK
)
/

PROMPT Adding PRIMARY Constraint To SPECIALIST_CONTRACTORSS Table

ALTER TABLE SPECIALIST_CONTRACTORSS ADD (
      PRIMARY KEY (SPEC_ID)
      CONSTRAINT SPECIALIST_PK
)
/

PROMPT Adding PRIMARY Constraint To SPECIALIST_SCHEDULE Table

ALTER TABLE SPECIALIST_SCHEDULE ADD (
      PRIMARY KEY (SPEC_ID1,
                   SCHE_ID2)
      CONSTRAINT SPECIALIST_SCHEDULE_PK
)
/

PROMPT Adding PRIMARY Constraint To SPECIAL_CUSTOMER Table

ALTER TABLE SPECIAL_CUSTOMER ADD (
      PRIMARY KEY (ID1,
                   CUST_ID2)
      CONSTRAINT SPECIAL_CUSTOMER_PK
)
/

PROMPT Adding PRIMARY Constraint To SPECIAL_NEEDS Table

ALTER TABLE SPECIAL_NEEDS ADD (
      PRIMARY KEY (ID)
      CONSTRAINT SPECIAL_NE_PK
)
/
```

```
PROMPT Adding PRIMARY Constraint To STREETS Table

ALTER TABLE STREETS ADD (
      PRIMARY KEY (STRE_ID)
      CONSTRAINT STREET_PK
)
/

PROMPT Adding PRIMARY Constraint To STREETWORKS Table

ALTER TABLE STREETWORKS ADD (
      PRIMARY KEY (ID)
      CONSTRAINT STREETWORK_PK
)
/

PROMPT Adding PRIMARY Constraint To STREETWORK_ORIG Table

ALTER TABLE STREETWORK_ORIG ADD (
      PRIMARY KEY (ID1,
                   ORIG_ID2)
      CONSTRAINT STREETWORK_ORIG_PK
)
/

PROMPT Adding PRIMARY Constraint To STREETWORK_SCHEDULE Table

ALTER TABLE STREETWORK_SCHEDULE ADD (
      PRIMARY KEY (ID1,
                   SCHE_ID2)
      CONSTRAINT STREETWORK_SCHEDULE_PK
)
/

PROMPT Adding PRIMARY Constraint To STREET_STREETWORK Table

ALTER TABLE STREET_STREETWORK ADD (
      PRIMARY KEY (STRE_ID1,
                   ID2)
      CONSTRAINT STREET_STREETWORK_PK
)
/

PROMPT Adding PRIMARY Constraint To TOWNS Table

ALTER TABLE TOWNS ADD (
      PRIMARY KEY (TOWN_ID)
      CONSTRAINT TOWN_PK
)
/

PROMPT Adding PRIMARY Constraint To USAGE Table

ALTER TABLE USAGE ADD (
      PRIMARY KEY (USAG_ID)
      CONSTRAINT USAGE_PK
)
/

PROMPT Adding PRIMARY Constraint To VEHICLES Table

ALTER TABLE VEHICLES ADD (
      PRIMARY KEY (VEHI_ID)
      CONSTRAINT VEHICLE_PK
)
/

PROMPT Adding PRIMARY Constraint To VEHICLE_USAGE Table

ALTER TABLE VEHICLE_USAGE ADD (
      PRIMARY KEY (VEHI_ID1,
                   USAG_ID2)
      CONSTRAINT VEHICLE_USAGE_PK
)
/
```

```
PROMPT Adding PRIMARY Constraint To VEHICULE_MAKERS Table

ALTER TABLE VEHICULE_MAKERS ADD (
      PRIMARY KEY (V_MA_ID)
      CONSTRAINT V_MAKE_PK
)
/

PROMPT Adding FOREIGN Constraint To ACCOUNTS Table

ALTER TABLE ACCOUNTS ADD (
      FOREIGN KEY (ACC_CUSTOMER_ID)
      REFERENCES  CUSTOMERS (
                   CUST_ID)
      CONSTRAINT ACC_SENT_TO
)
/

PROMPT Adding FOREIGN Constraint To APPLIANCES Table

ALTER TABLE APPLIANCES ADD (
      FOREIGN KEY (APPL_CUSTOMER_ID)
      REFERENCES  CUSTOMERS (
                   CUST_ID)
      CONSTRAINT APPL_BELONG_TO
)
/

PROMPT Adding FOREIGN Constraint To APPLIANCES Table

ALTER TABLE APPLIANCES ADD (
      FOREIGN KEY (APPL_APL_T_ID)
      REFERENCES  APPLIANCE_TYPES (
                   APL__ID)
      CONSTRAINT APPL_HAS
)
/

PROMPT Adding FOREIGN Constraint To APPLIANCES Table

ALTER TABLE APPLIANCES ADD (
      FOREIGN KEY (APPL_APL_M_ID)
      REFERENCES  APPLIANCE_MAKES (
                   ID)
      CONSTRAINT APPL_MADE_BY
)
/

PROMPT Adding FOREIGN Constraint To BUILDINGS Table

ALTER TABLE BUILDINGS ADD (
      FOREIGN KEY (BUIL_STREET_ID)
      REFERENCES  STREETS (
                   STRE_ID)
      CONSTRAINT BUILD_SITUATED_ON
)
/

PROMPT Adding FOREIGN Constraint To BUILDINGS Table

ALTER TABLE BUILDINGS ADD (
      FOREIGN KEY (BUIL_CUSTOMER_ID)
      REFERENCES  CUSTOMERS (
                   CUST_ID)
      CONSTRAINT BUILD_MAY_CONTAIN
)
/

PROMPT Adding FOREIGN Constraint To BUILDINGS Table

ALTER TABLE BUILDINGS ADD (
      FOREIGN KEY (BUIL_HOUSEHOLD_ID)
      REFERENCES  HOUSEHOLDS (
                   HOUS_ID)
      CONSTRAINT BUILD_MAY_CONTAIN2
)
/
```

```
PROMPT Adding FOREIGN Constraint To BUILDINGS Table

ALTER TABLE BUILDINGS ADD (
      FOREIGN KEY (BUIL_CENSUS_ID)
      REFERENCES  CENSUS (
                   CENS_ID)
      CONSTRAINT BUILD_HAVE
)
/

PROMPT Adding FOREIGN Constraint To BUILDINGS Table

ALTER TABLE BUILDINGS ADD (
      FOREIGN KEY (BUIL_CARRIER_ID)
      REFERENCES  CARRIERS (
                   CARR_ID)
      CONSTRAINT BUILD_CONTAINS
)
/

PROMPT Adding FOREIGN Constraint To BUILDINGS Table

ALTER TABLE BUILDINGS ADD (
      FOREIGN KEY (BUIL_SPECIALIST_ID)
      REFERENCES  SPECIALIST_CONTRACTORSS (
                   SPEC_ID)
      CONSTRAINT BUILD_CONTAINS2
)
/

PROMPT Adding FOREIGN Constraint To BUILDINGS Table

ALTER TABLE BUILDINGS ADD (
      FOREIGN KEY (BUIL_EMPLOYEE_ID)
      REFERENCES  EMPLOYEES (
                   EMPL_ID)
      CONSTRAINT BUILD_CONTAIN
)
/

PROMPT Adding FOREIGN Constraint To BUILD_BUILDU Table

ALTER TABLE BUILD_BUILDU ADD (
      FOREIGN KEY (BUIL_ID1)
      REFERENCES  BUILDINGS (
                   BUIL_ID)
      CONSTRAINT BUIL_BUIL_FK
)
/

PROMPT Adding FOREIGN Constraint To BUILD_BUILDU Table

ALTER TABLE BUILD_BUILDU ADD (
      FOREIGN KEY (ID2)
      REFERENCES  BUILDING_USES (
                   ID)
      CONSTRAINT BUIL_BUIL_FK2
)
/

PROMPT Adding FOREIGN Constraint To CARRIERS Table

ALTER TABLE CARRIERS ADD (
      FOREIGN KEY (CARR_BUILD_ID)
      REFERENCES  BUILDINGS (
                   BUIL_ID)
      CONSTRAINT CARRIER_RESIDENT_IN
)
/

PROMPT Adding FOREIGN Constraint To CARRIER_PIPE Table

ALTER TABLE CARRIER_PIPE ADD (
      FOREIGN KEY (CARR_ID1)
      REFERENCES  CARRIERS (
                   CARR_ID)
      CONSTRAINT CARR_PIPE_FK
)
/

PROMPT Adding FOREIGN Constraint To CARRIER_PIPE Table

ALTER TABLE CARRIER_PIPE ADD (
      FOREIGN KEY (PIPE_RECORD_ID2)
      REFERENCES  PIPES (
                  PIPE_RECORD_ID)
      CONSTRAINT CARR_PIPE_FK2
)
/

PROMPT Adding FOREIGN Constraint To CENSUS Table

ALTER TABLE CENSUS ADD (
      FOREIGN KEY (CENS_BUILD_ID)
      REFERENCES  BUILDINGS (
                  BUIL_ID)
      CONSTRAINT CENSUS_RECORD
)
/

PROMPT Adding FOREIGN Constraint To COMPLAINTS_CUSTOMER Table

ALTER TABLE COMPLAINTS_CUSTOMER ADD (
      FOREIGN KEY (COMP_ID1)
      REFERENCES  COMPLAINTS (
                  COMP_ID)
      CONSTRAINT COMP_CUST_FK
)
/

PROMPT Adding FOREIGN Constraint To COMPLAINTS_CUSTOMER Table

ALTER TABLE COMPLAINTS_CUSTOMER ADD (
      FOREIGN KEY (CUST_ID2)
      REFERENCES  CUSTOMERS (
                  CUST_ID)
      CONSTRAINT COMP_CUST_FK2
)
/

PROMPT Adding FOREIGN Constraint To COMPLETIONS Table

ALTER TABLE COMPLETIONS ADD (
      FOREIGN KEY (STREETWORK_ID)
      REFERENCES  STREETWORKS (
                  ID)
      CONSTRAINT COMPLETION_HAVE
)
/

PROMPT Adding FOREIGN Constraint To COMP_APL_M Table

ALTER TABLE COMP_APL_M ADD (
      FOREIGN KEY (ID1)
      REFERENCES  COMPONENTS (
                  ID)
      CONSTRAINT COMP_APL__FK3
)
/

PROMPT Adding FOREIGN Constraint To COMP_APL_M Table

ALTER TABLE COMP_APL_M ADD (
      FOREIGN KEY (ID2)
      REFERENCES  APPLIANCE_MAKES (
                  ID)
      CONSTRAINT COMP_APL__FK4
)
/

PROMPT Adding FOREIGN Constraint To COMP_APL_T Table

ALTER TABLE COMP_APL_T ADD (
      FOREIGN KEY (ID1)
      REFERENCES  COMPONENTS (
                  ID)
      CONSTRAINT COMP_APL__FK
)
/

PROMPT Adding FOREIGN Constraint To COMP_APL_T Table

ALTER TABLE COMP_APL_T ADD (
      FOREIGN KEY (APL__ID2)
      REFERENCES  APPLIANCE_TYPES (
                  APL__ID)
      CONSTRAINT COMP_APL__FK2
)
/

PROMPT Adding FOREIGN Constraint To COUNCIL_TAX Table

ALTER TABLE COUNCIL_TAX ADD (
      FOREIGN KEY (COUN_HOUSEHOLD_ID)
      REFERENCES  HOUSEHOLDS (
                  HOUS_ID)
      CONSTRAINT COUNCIL_OWED_BY
)
/

PROMPT Adding FOREIGN Constraint To COUNCIL_TAX Table

ALTER TABLE COUNCIL_TAX ADD (
      FOREIGN KEY (COUN_CHARGE_ID)
      REFERENCES  CHARGES (
                  CHAR_ID)
      CONSTRAINT COUNCIL_HAVE
)
/

PROMPT Adding FOREIGN Constraint To CUSTOMERS Table

ALTER TABLE CUSTOMERS ADD (
      FOREIGN KEY (CUST_BUILD_ID)
      REFERENCES  BUILDINGS (
                  BUIL_ID)
      CONSTRAINT CUSTOMER_RESIDES_IN
)
/

PROMPT Adding FOREIGN Constraint To CUSTOMERS Table

ALTER TABLE CUSTOMERS ADD (
      FOREIGN KEY (CUST_PIPE_RECORD_ID)
      REFERENCES  PIPES (
                  PIPE_RECORD_ID)
      CONSTRAINT CUSTOMER_CONNECTED_TO
)
/

PROMPT Adding FOREIGN Constraint To CUSTOMERS Table

ALTER TABLE CUSTOMERS ADD (
      FOREIGN KEY (CUST_EMPLOYEE_ID)
      REFERENCES  EMPLOYEES (
                  EMPL_ID)
      CONSTRAINT CUSTOMER_BE
)
/

PROMPT Adding FOREIGN Constraint To EMERGENCY_CUSTOMER Table

ALTER TABLE EMERGENCY_CUSTOMER ADD (
      FOREIGN KEY (EMER_ID1)
      REFERENCES  EMERGENCIES (
                  EMER_ID)
      CONSTRAINT EMER_CUST_FK
)
/
```

```
PROMPT Adding FOREIGN Constraint To EMERGENCY_CUSTOMER Table

ALTER TABLE EMERGENCY_CUSTOMER ADD (
      FOREIGN KEY (CUST_ID2)
      REFERENCES  CUSTOMERS (
                  CUST_ID)
      CONSTRAINT EMER_CUST_FK2
)
/

PROMPT Adding FOREIGN Constraint To EMPLOYEES Table

ALTER TABLE EMPLOYEES ADD (
      FOREIGN KEY (EMPL_CUSTOMER_ID)
      REFERENCES  CUSTOMERS (
                  CUST_ID)
      CONSTRAINT EMPLOYEE_BE
)
/

PROMPT Adding FOREIGN Constraint To EMPLOYEES Table

ALTER TABLE EMPLOYEES ADD (
      FOREIGN KEY (EMPL_PAY_ID)
      REFERENCES  PAY (
                  PAY_ID)
      CONSTRAINT EMPLOYEE_RECEIVE
)
/

PROMPT Adding FOREIGN Constraint To EMPLOYEES Table

ALTER TABLE EMPLOYEES ADD (
      FOREIGN KEY (EMPL_SCHEDULE_ID)
      REFERENCES  SCHEDULES (
                  SCHE_ID)
      CONSTRAINT EMPLOYEE_SCHEDULED
)
/

PROMPT Adding FOREIGN Constraint To GOVERNOR_STATIONS Table

ALTER TABLE GOVERNOR_STATIONS ADD (
      FOREIGN KEY (GOVE_PIPE_RECORD_ID)
      REFERENCES  PIPES (
                  PIPE_RECORD_ID)
      CONSTRAINT GOVERNOR_SITUATED_ON
)
/

PROMPT Adding FOREIGN Constraint To HOUSEHOLDS Table

ALTER TABLE HOUSEHOLDS ADD (
      FOREIGN KEY (HOUS_BUILD_ID)
      REFERENCES  BUILDINGS (
                  BUIL_ID)
      CONSTRAINT HOUSEHOLD_RESIDES_IN
)
/

PROMPT Adding FOREIGN Constraint To HOUSEHOLDS Table

ALTER TABLE HOUSEHOLDS ADD (
      FOREIGN KEY (HOUS_COUNCIL_T_ID)
      REFERENCES  COUNCIL_TAX (
                  COUN_ID)
      CONSTRAINT HOUSEHOLD_PAYS
)
/

PROMPT Adding FOREIGN Constraint To INJECTION_POINTS Table

ALTER TABLE INJECTION_POINTS ADD (
      FOREIGN KEY (INJE_PIPE_RECORD_ID)
      REFERENCES  PIPES (
                  PIPE_RECORD_ID)
      CONSTRAINT INJECTION_SITUATED_ON
```

```
)
/

PROMPT Adding FOREIGN Constraint To INSTALLATIONS Table

ALTER TABLE INSTALLATIONS ADD (
      FOREIGN KEY (INST_CUSTOMER_ID)
      REFERENCES  CUSTOMERS (
                   CUST_ID)
      CONSTRAINT INSTALLATI_APPLY_TO
)
/

PROMPT Adding FOREIGN Constraint To INSTALL_APPL Table

ALTER TABLE INSTALL_APPL ADD (
      FOREIGN KEY (INST_ID1)
      REFERENCES  INSTALLATIONS (
                   INST_ID)
      CONSTRAINT INST_APPL_FK
)
/

PROMPT Adding FOREIGN Constraint To INSTALL_APPL Table

ALTER TABLE INSTALL_APPL ADD (
      FOREIGN KEY (APPL_ID2)
      REFERENCES  APPLIANCES (
                   APPL_ID)
      CONSTRAINT INST_APPL_FK2
)
/

PROMPT Adding FOREIGN Constraint To METERS Table

ALTER TABLE METERS ADD (
      FOREIGN KEY (METE_CUSTOMER_ID)
      REFERENCES  CUSTOMERS (
                   CUST_ID)
      CONSTRAINT METER_APPLY_TO
)
/

PROMPT Adding FOREIGN Constraint To METERS Table

ALTER TABLE METERS ADD (
      FOREIGN KEY (METE_METER_T_ID)
      REFERENCES  METER_TYPES (
                   ID)
      CONSTRAINT METER_HAS
)
/

PROMPT Adding FOREIGN Constraint To PAY Table

ALTER TABLE PAY ADD (
      FOREIGN KEY (PAY_EMPLOYEE_ID)
      REFERENCES  EMPLOYEES (
                   EMPL_ID)
      CONSTRAINT PAY_PAID_TO
)
/

PROMPT Adding FOREIGN Constraint To PIPES Table

ALTER TABLE PIPES ADD (
      FOREIGN KEY (PIPE_PIPE_NETWO_ID)
      REFERENCES  PIPE_NETWORKS (
                   ID)
      CONSTRAINT PIPE_PART_OF
)
/

PROMPT Adding FOREIGN Constraint To PIPES Table

ALTER TABLE PIPES ADD (
      FOREIGN KEY (PIPE_GOVERNOR_ID)
      REFERENCES  GOVERNOR_STATIONS (
                   GOVE_ID)
      CONSTRAINT PIPE_SUPPORT
)
/

PROMPT Adding FOREIGN Constraint To PIPE_STREET Table

ALTER TABLE PIPE_STREET ADD (
      FOREIGN KEY (PIPE_RECORD_ID1)
      REFERENCES  PIPES (
                   PIPE_RECORD_ID)
      CONSTRAINT PIPE_STRE_FK
)
/

PROMPT Adding FOREIGN Constraint To PIPE_STREET Table

ALTER TABLE PIPE_STREET ADD (
      FOREIGN KEY (STRE_ID2)
      REFERENCES  STREETS (
                   STRE_ID)
      CONSTRAINT PIPE_STRE_FK2
)
/

PROMPT Adding FOREIGN Constraint To PLANS Table

ALTER TABLE PLANS ADD (
      FOREIGN KEY (PLAN_STREETWORK_ID)
      REFERENCES  STREETWORKS (
                   ID)
      CONSTRAINT PLANS_PLANNED
)
/

PROMPT Adding FOREIGN Constraint To SCHEDULES Table

ALTER TABLE SCHEDULES ADD (
      FOREIGN KEY (SCHE_EMPLOYEE_ID)
      REFERENCES  EMPLOYEES (
                   EMPL_ID)
      CONSTRAINT SCHEDULE_RECEIVED
)
/

PROMPT Adding FOREIGN Constraint To SERVICE_APPL Table

ALTER TABLE SERVICE_APPL ADD (
      FOREIGN KEY (SERV_ID1)
      REFERENCES  SERVICE_CONTRACTS (
                   SERV_ID)
      CONSTRAINT SERV_APPL_FK
)
/

PROMPT Adding FOREIGN Constraint To SERVICE_APPL Table

ALTER TABLE SERVICE_APPL ADD (
      FOREIGN KEY (APPL_ID2)
      REFERENCES  APPLIANCES (
                   APPL_ID)
      CONSTRAINT SERV_APPL_FK2
)
/

PROMPT Adding FOREIGN Constraint To SERVICE_CONTRACTS Table

ALTER TABLE SERVICE_CONTRACTS ADD (
      FOREIGN KEY (SERV_CUSTOMER_ID)
      REFERENCES  CUSTOMERS (
                   CUST_ID)
      CONSTRAINT SERVICE_CO_PERTAIN_TO
)
/

PROMPT Adding FOREIGN Constraint To SIPHONS Table
```

```
ALTER TABLE SIPHONS ADD (
      FOREIGN KEY (SIPH_PIPE_RECORD_ID)
      REFERENCES  PIPES (
                   PIPE_RECORD_ID)
      CONSTRAINT SIPHON_SITUATED_ON
)
/

PROMPT Adding FOREIGN Constraint To SPECIALIST_CONTRACTORSS Table

ALTER TABLE SPECIALIST_CONTRACTORSS ADD (
      FOREIGN KEY (SPEC_BUILD_ID)
      REFERENCES  BUILDINGS (
                   BUIL_ID)
      CONSTRAINT SPECIALIST_RESIDE_IN
)
/

PROMPT Adding FOREIGN Constraint To SPECIALIST_SCHEDULE Table

ALTER TABLE SPECIALIST_SCHEDULE ADD (
      FOREIGN KEY (SPEC_ID1)
      REFERENCES  SPECIALIST_CONTRACTORSS (
                   SPEC_ID)
      CONSTRAINT SPEC_SCHE_FK
)
/

PROMPT Adding FOREIGN Constraint To SPECIALIST_SCHEDULE Table

ALTER TABLE SPECIALIST_SCHEDULE ADD (
      FOREIGN KEY (SCHE_ID2)
      REFERENCES  SCHEDULES (
                   SCHE_ID)
      CONSTRAINT SPEC_SCHE_FK2
)
/

PROMPT Adding FOREIGN Constraint To SPECIAL_CUSTOMER Table

ALTER TABLE SPECIAL_CUSTOMER ADD (
      FOREIGN KEY (ID1)
      REFERENCES  SPECIAL_NEEDS (
                   ID)
      CONSTRAINT SPEC_CUST_FK
)
/

PROMPT Adding FOREIGN Constraint To SPECIAL_CUSTOMER Table

ALTER TABLE SPECIAL_CUSTOMER ADD (
      FOREIGN KEY (CUST_ID2)
      REFERENCES  CUSTOMERS (
                   CUST_ID)
      CONSTRAINT SPEC_CUST_FK2
)
/

PROMPT Adding FOREIGN Constraint To STREETS Table

ALTER TABLE STREETS ADD (
      FOREIGN KEY (STRE_TOWN_ID)
      REFERENCES  TOWNS (
                   TOWN_ID)
      CONSTRAINT STREET_LOCATED_IN
)
/

PROMPT Adding FOREIGN Constraint To STREETWORK_ORIG Table

ALTER TABLE STREETWORK_ORIG ADD (
      FOREIGN KEY (ID1)
      REFERENCES  STREETWORKS (
                   ID)
      CONSTRAINT STRE_ORIG_FK
)
```

```
/

PROMPT Adding FOREIGN Constraint To STREETWORK_ORIG Table

ALTER TABLE STREETWORK_ORIG ADD (
      FOREIGN KEY (ORIG_ID2)
      REFERENCES  ORIGINATOR_RECIPIENTS (
                  ORIG_ID)
      CONSTRAINT STRE_ORIG_FK2
)
/

PROMPT Adding FOREIGN Constraint To STREETWORK_SCHEDULE Table

ALTER TABLE STREETWORK_SCHEDULE ADD (
      FOREIGN KEY (ID1)
      REFERENCES  STREETWORKS (
                  ID)
      CONSTRAINT STRE_SCHE_FK
)
/

PROMPT Adding FOREIGN Constraint To STREETWORK_SCHEDULE Table

ALTER TABLE STREETWORK_SCHEDULE ADD (
      FOREIGN KEY (SCHE_ID2)
      REFERENCES  SCHEDULES (
                  SCHE_ID)
      CONSTRAINT STRE_SCHE_FK2
)
/

PROMPT Adding FOREIGN Constraint To STREET_STREETWORK Table

ALTER TABLE STREET_STREETWORK ADD (
      FOREIGN KEY (STRE_ID1)
      REFERENCES  STREETS (
                  STRE_ID)
      CONSTRAINT STRE_STRE_FK
)
/

PROMPT Adding FOREIGN Constraint To STREET_STREETWORK Table

ALTER TABLE STREET_STREETWORK ADD (
      FOREIGN KEY (ID2)
      REFERENCES  STREETWORKS (
                  ID)
      CONSTRAINT STRE_STRE_FK2
)
/

PROMPT Adding FOREIGN Constraint To TOWNS Table

ALTER TABLE TOWNS ADD (
      FOREIGN KEY (TOWN_COUNTY_ID)
      REFERENCES  COUNTIES (
                  ID)
      CONSTRAINT TOWN_LOCATED_IN
)
/

PROMPT Adding FOREIGN Constraint To VEHICLES Table

ALTER TABLE VEHICLES ADD (
      FOREIGN KEY (VEHI_SCHEDULE_ID)
      REFERENCES  SCHEDULES (
                  SCHE_ID)
      CONSTRAINT VEHICLE_SCHEDULED
)
/

PROMPT Adding FOREIGN Constraint To VEHICLES Table

ALTER TABLE VEHICLES ADD (
      FOREIGN KEY (VEHI_V_MAKE_ID)
      REFERENCES  VEHICULE_MAKERS (
                   V_MA_ID)
      CONSTRAINT VEHICLE_MADE_BY
)
/

PROMPT Adding FOREIGN Constraint To VEHICLE_USAGE Table

ALTER TABLE VEHICLE_USAGE ADD (
      FOREIGN KEY (VEHI_ID1)
      REFERENCES  VEHICLES (
                   VEHI_ID)
      CONSTRAINT VEHI_USAG_FK
)
/

PROMPT Adding FOREIGN Constraint To VEHICLE_USAGE Table

ALTER TABLE VEHICLE_USAGE ADD (
      FOREIGN KEY (USAG_ID2)
      REFERENCES  USAGE (
                   USAG_ID)
      CONSTRAINT VEHI_USAG_FK2
)
/

REM
REM  End of command file
REM
EXIT
```

Appendix C

The following is a validity check report from CASE*Designer, and was use to check mappings between entities

```
CASE*Dictionary Reports

         Report          : ENTITY MODEL INFORMATION

         Filename        : cdents.lis

         Run By          : OPS$MJT

         Report date     : 17-APR-98 14:22:35


         +------------------------------------------------------+
         |                                                      |
         | Parameter values                                     |
         |                                                      |
         | Application System : GAS                             |
         | Version            : 1                               |
         | Page Numbers ?     : Y                               |
         | 4 Character Prefix :                                 |
         |                                                      |
         +------------------------------------------------------+
ACCOUNT

Each ACCOUNT has significance as Collates all information necessary for creating
bills to send to customers

Information about ACCOUNT includes id, credit_rating, payment  due,  start_date,
aprox_annual_usage, end_date, etc.

Each ACCOUNT
     must be sent to one and only one CUSTOMER


APPLIANCE

Each APPLIANCE has significance as Stores all gas appliance types available

Information about APPLIANCE includes id, make, description, type, etc.

Each APPLIANCE
     must be made by one and only one APPLIANCE MAKER
 and must be has one and only one APPLIANCE TYPE
 and may be belong to one and only one CUSTOMER
 and may be installed one or more INSTALLATIONS
 and may be belong to one or more SERVICE CONTRACTS


APPLIANCE MAKER

Each APPLIANCE MAKER

Information about APPLIANCE MAKER includes id, name, etc.

Each APPLIANCE MAKER
     may be make one or more APPLIANCES
 and may be require one or more COMPONENTS


APPLIANCE TYPE

Each APPLIANCE TYPE

Information about APPLIANCE TYPE includes id, name, etc.

Each APPLIANCE TYPE
     may be belongs one or more APPLIANCES
 and may be require one or more COMPONENTS
```

AVENUE

Each AVENUE

Each AVENUE
    may be contains one or more HOMES

BUILDING

Each BUILDING has significance as Description of buildings with gas supplies

Information about BUILDING includes id, description, category, label_point, xcoord, ycoord, number, postcode, name, etc.

Each BUILDING
    must be may contain one and only one HOUSEHOLD
 and may be have one or more BUILDING USES
 and may be contains one and only one CARRIER
 and may be have one and only one CENSUS
 and may be may contain one and only one CUSTOMER
 and may be contain one and only one EMPLOYEE
 and may be contains one and only one SPECIALIST CONTRACTOR
 and may be situated on one and only one STREET

BUILDING USE

Each BUILDING USE also known as BUILD USE, has significance as Describes the type of building use eg private, domestic etc

Information about BUILDING USE includes id, use, etc.

Each BUILDING USE
    may be have one or more BUILDINGS

CARRIER

Each CARRIER has significance as details of carriers supplying gas into the network

Information about CARRIER includes id, name, telephone, contact name, etc.

Each CARRIER
    may be resident in one and only one BUILDING
 and may be inserts into one or more PIPES

CENSUS

Each CENSUS has significance as contains census data from 1991 census

Information about CENSUS includes id, etc.

Each CENSUS
must be record one and only one BUILDING

CHARGE

Each CHARGE has significance as contains band charges for council tax

Information about CHARGE includes id, rate, etc.

Each CHARGE
may be used by one or more COUNCIL TAX

COMPLAINTS

Each COMPLAINTS has significance as description of complaint received from customers

Information about COMPLAINTS includes id, description, action, etc.

Each COMPLAINTS
may be come from one or more CUSTOMERS

COMPLETION

Each COMPLETION has significance as completeion details of streetworks

Information about COMPLETION includes id, description, completion_date, etc.

Each COMPLETION
must be have one and only one STREETWORKS

COMPONENTS

Each COMPONENTS

Information about COMPONENTS includes id, description, etc.

continued on next page

```
COMPONENTS continued

Each COMPONENTS
     may be fit one or more APPLIANCE MAKERS
 and may be fit one or more APPLIANCE TYPES


COUNCIL TAX

Each COUNCIL TAX has significance as council tax details

Information about COUNCIL TAX includes id,  band,  amount  paid,  last  payment,
council name, etc.

Each COUNCIL TAX
     must be have one and only one CHARGE
 and may be owed by one and only one HOUSEHOLD


COUNTY

Each COUNTY has significance as Name of county, shire or region

Information about COUNTY includes id, name, etc.

Each COUNTY
     may be contain one or more TOWNS


CUSTOMER

Each CUSTOMER has significance as customer details

Information about CUSTOMER includes id, surname, initials, title, etc.

Each CUSTOMER
     must be owns one or more APPLIANCES
 and may be receive one or more ACCOUNTS
 and may be resides in one and only one BUILDING
 and may be make one or more COMPLAINTS
 and may be report one or more EMERGENCIES
 and may be be one and only one EMPLOYEE
 and may be require one or more INSTALLATIONS
 and may be have one or more METERS
 and may be connected to one and only one PIPE
 and may be draw up one or more SERVICE CONTRACTS
 and may be have one or more SPECIAL NEEDS
```

EMERGENCY

Each EMERGENCY

Information about EMERGENCY includes id, description, action, priority, etc.

Each EMERGENCY
may be occur to one or more CUSTOMERS

EMPLOYEE

Each EMPLOYEE

Information about EMPLOYEE includes id, surname, initials, title, department, status, job_description, etc.

Each EMPLOYEE
may be resides in one or more BUILDINGS
and may be be one and only one CUSTOMER
and may be receive one and only one PAY
and may be scheduled one and only one SCHEDULE

GOVERNOR STATION

Each GOVERNOR STATION

Information about GOVERNOR STATION includes id, settings, date_of_service, description, etc.

Each GOVERNOR STATION
must be situated on one and only one PIPE

HOME

Each HOME

Each HOME
must be situated on one and only one AVENUE

HOUSEHOLD

Each HOUSEHOLD

Information about HOUSEHOLD includes id, surname, street, town, county, postcode, type, etc.

continued on next page

HOUSEHOLD continued

Each HOUSEHOLD
    must be pays one and only one COUNCIL TAX
 and may be resides in one and only one BUILDING

INJECTION POINT

Each INJECTION POINT

Information about INJECTION POINT includes id, settings, date, description, etc.

Each INJECTION POINT
    must be situated on one and only one PIPE

INSTALLATION

Each INSTALLATION

Information about INSTALLATION includes id, description, cost, start_date, end_date, start_time, end_time, etc.

Each INSTALLATION
    must be install one or more APPLIANCES
 and must be apply to one and only one CUSTOMER

LAND PARCEL

Each LAND PARCEL

LAND USE ZONE

Each LAND USE ZONE

Each LAND USE ZONE
    may be has one or more PERMITS

METER

Each METER

Information about METER includes id, reading, date, time, etc.

continued on next page

METER continued

Each METER
    must be apply to one and only one CUSTOMER
 and must be has one and only one METER TYPE

METER TYPE

Each METER TYPE

Information about METER TYPE includes id, type, etc.

Each METER TYPE
    may be belong one or more METERS

ORIGINATOR_RECIPIENT

Each ORIGINATOR_RECIPIENT

Information about ORIGINATOR_RECIPIENT includes id, name, etc.

Each ORIGINATOR_RECIPIENT
    may be originate/ receive one or more STREETWORKS

OWNER

Each OWNER

PAY

Each PAY

Information about PAY includes id, rate, etc.

Each PAY
    may be paid to one and only one EMPLOYEE

PERMIT

Each PERMIT

Each PERMIT
    must be has one and only one LAND USE ZONE

PIPE

Each PIPE

Information about PIPE includes record_id, dn1, action_type, pressure, main_id, status, length, method_laid, date_laid, points_rating, protection, district, street_id, diameter, diameter_units, material, sdr, joint_type, comments, mtr, etc.

```
Each PIPE
     must be carries to one or more CUSTOMERS
 and may be carries for one or more CARRIERS
 and may be support one and only one GOVERNOR STATION
 and may be support one or more INJECTION POINTS
 and may be part of one and only one PIPE NETWORK
 and may be support one or more SIPHONS
 and may be refered to one or more STREETS
```

PIPE NETWORK

Each PIPE NETWORK

Information about PIPE NETWORK includes id, name, total_length, connectivity_details, etc.

```
Each PIPE NETWORK
     must be comprises one or more PIPES
```

PLANS

Each PLANS

Information about PLANS includes id, description, blue_print, etc.

```
Each PLANS
     must be planned one and only one STREETWORKS
```

SCHEDULE

Each SCHEDULE

Information about SCHEDULE includes id, etc.

```
Each SCHEDULE
     may be received one and only one EMPLOYEE
 and may be required one or more SPECIALIST CONTRACTORS
 and may be schedule one or more STREETWORKS
 and may be required one or more VEHICLES
```

SERVICE CONTRACTS

Each SERVICE CONTRACTS

Information about SERVICE CONTRACTS includes id, description, start_date, cost, end_date, start_time, end_time, etc.

Each SERVICE CONTRACTS
must be contain one or more APPLIANCES
and must be pertain to one and only one CUSTOMER

SIPHON

Each SIPHON

Information about SIPHON includes id, settings, date, description, photo, etc.

Each SIPHON
must be situated on one and only one PIPE

SPECIAL NEEDS

Each SPECIAL NEEDS

Information about SPECIAL NEEDS includes id, type, priority, etc.

Each SPECIAL NEEDS
may be affect one or more CUSTOMERS

SPECIALIST CONTRACTOR

Each SPECIALIST CONTRACTOR

Information about SPECIALIST CONTRACTOR includes id, name, contact_name, telephone, etc.

Each SPECIALIST CONTRACTOR
may be reside in one and only one BUILDING
and may be contracted one or more SCHEDULES

STREET

Each STREET

Information about STREET includes id, name, type, route_number, etc.

continued on next page

STREET continued

Each STREET
must be contains one or more BUILDINGS
and may be refer one or more PIPES
and may be may contain one or more STREETWORKS
and may be located in one and only one TOWN

STREETWORKS

Each STREETWORKS

Information about STREETWORKS includes id, reinstatement, notice_period, description, start_date, end_date, start_time, end_time, orig_cross_ref, recip_cross_ref, comments, etc.

Each STREETWORKS
may be have one or more COMPLETIONS
and may be have one or more ORIGINATOR_RECIPIENTS
and may be have one or more PLANS
and may be scheduled one or more SCHEDULES
and may be situated on one or more STREETS

TOWN

Each TOWN

Information about TOWN includes id, name, etc.

Each TOWN
must be located in one and only one COUNTY
and may be contain one or more STREETS

USAGE

Each USAGE

Information about USAGE includes id, description, etc.

Each USAGE
may be apply one or more VEHICLES

VEHICLE

Each VEHICLE

Information about VEHICLE includes id, number_plate, description, comments, etc.

continued on next page

VEHICLE continued

Each VEHICLE
must be scheduled one and only one SCHEDULE
and must be made by one and only one VEHICULE MAKER
and may be has one or more USAGE

VEHICULE MAKER

Each VEHICULE MAKER

Information about VEHICULE MAKER includes name, id, etc.

Each VEHICULE MAKER
may be make one or more VEHICLES

CASE*Dictionary Reports

ENTITY MODEL INFORMATION

End of Report

Appendix D

Example of C code used to create data for the corporate databes. This code is used to create names for customers and employees in the corporate database.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAX_SURNAMES 2926
#define NUMBER_REQUID 1500000
char Surname[MAX_SURNAMES][80];

struct Initial {
      char* letter;
      int  weight;
} InitTab[] = {
      " ", -1,
      "A.", 5922,"B.", 10528,"C.", 15134,"D.", 20398,
      "E.", 26978,"F.", 32242,"G.", 36190,"H.", 40138,
      "I.", 46060,"J.", 49350,"K.", 52640,"L.", 57246,
      "M.", 61194,"N.", 65800,"O.", 70406,"P.", 74354,
      "Q.", 75670,"R.", 80276,"S.", 85540,"T.", 90146,
      "U.", 92120,"V.", 94094,"W.", 98042,"X.", 98700,
      "Y.", 100674,"Z.", 101332 };

extern int random();
char *addstr(d,s)
char *d, *s;
{
      while((*d++=*s++)!='\0');
      return --d;
}


char * GetSurname()
{
      char Name[400], *p;
      int i, l1,l2;
      p = Name;
        p = addstr(p," ");
      i = random (MAX_SURNAMES);
      p = addstr(p,Surname[i]);
#ifdef DEBUG
      fprintf(stderr,"Surname %s\n",Name);
#endif
      if (random(1000)==1){
            i = random(MAX_SURNAMES);
            l1 = strlen(Name);
            l2 = strlen(Surname[i]) +1;
            if ((l1 +l2 ) <400 ){
                  p = addstr(p,"-");
                  i = random(MAX_SURNAMES);
                  p = addstr(p,Surname[i]);
#ifdef DEBUG
                  fprintf(stderr,"Surname %s\n",Name);
#endif
```

```
            }
      }
      return Name;
}

char * MakeTitle()
{
      char Title[10];
      int i;
      i = random(1000);
      if (i < 450) strcpy(Title, "Mr. ");
      else
          if (i < 455) strcpy(Title, "Prof. ");
          else
            if (i< 700) strcpy(Title, "Ms. ");
            else
                if (i<965) strcpy(Title, "Mrs. ");
                else
                  strcpy(Title, "Dr. ");
#ifdef DEBUG
      fprintf(stderr,"Title %s\n",Title);
#endif
      return Title;
}


char * GetInitial(weight)
int weight;
{
      int i=0;
/*    printf("%d\n",weight);  */
      while ((InitTab[i].weight < weight)&& (i < 26))
            i++;
/*    printf("%d \t%s\n",i,InitTab[i].letter);*/
      return (InitTab[i].letter);
}

char * GenerateInitials()
{
      int i, count, rnum=0;
      char *p, Inits[50];
      p = Inits;

      i=random(100);
      count =3;
      if (i<75) count--;
      if (i<25) count--;

      for (i=0; i< count; i++){
            rnum = random(101331);
            p = addstr(p, GetInitial(rnum));

      }

#ifdef DEBUG
      fprintf(stderr,"Initials %s\n",Inits);
#endif
      return(Inits);
}
```

```
main()
{
        int i;
        FILE *fopen(), *fin, *fout;
        char INFILE[80], OUTFILE[80];
        char buffer[512], *res, *p, *pos;

        res = buffer;


#define ERRORMSG "Error opening file : %s.\n"
        strcpy(INFILE, "/home/mette/oracle1/datamodel/fred2.lis");
        strcpy(OUTFILE,
"/home/mette/oracle1/datamodel/names/names.dat");
        if ((fin=fopen(INFILE,"r"))==NULL){
                printf(ERRORMSG, INFILE);
                exit(1);
        }
#ifndef DEBUG
        if ((fout = fopen(OUTFILE, "w"))==NULL){
                printf(ERRORMSG, OUTFILE);
                exit(1);
        }
#endif
        for (i=0; i< MAX_SURNAMES; i++){
                fscanf(fin,"%s",Surname[i]);
        }
        for (i=0; i< NUMBER_REQUID; i++){
#ifdef DEBUG
        fprintf(stderr,"Record  %d\n",i);
#endif
             res = addstr( res, MakeTitle());
             res = addstr( res , GenerateInitials());
             res =addstr(res, GetSurname());

             fprintf(fout,"%s\n",buffer);

#ifdef DEBUG
        fprintf(stderr,"Name %s\n",buffer);
#endif
        if (!( i %1000 )) fprintf(stderr,"Name \t%d\n",i);
        strcpy (buffer,"");
        res=buffer;
        }
}
```

Appendix E

Examples of Smallworld magik code used to create the entities and structures necessary to digitise the gas mains network.

Gas_RWO.MAGIK

```
make_rwo_init ()
$
define_manifold(:gas_network,1)
$

# define other point objects for the database

define_rwo_table(!current_dsview!,:siphon,
      vec(vec(:siphon_id,:ds_uint)),
      1,1004,"rwo.ds")
$

define_application_code(:siphon,:location,1,:point,:gas_network)
$

define_rwo_table(!current_dsview!,:pressure_point,
      vec(vec(:pressure_point_id,:ds_uint)),
      1,1005,"rwo.ds")
$

define_application_code(:pressure_point,:location,1,:point,:gas_netwo
rk)
$

define_rwo_table(!current_dsview!,:carbo_seal,
      vec(vec(:carbo_seal_id,:ds_uint)),
      1,1006,"rwo.ds")
$

define_application_code(:carbo_seal,:location,1,:point,:gas_network)
$

define_rwo_table(!current_dsview!,:govenor_houses,
      vec(vec(:govenor_houses,:ds_uint)),
      1,1007,"rwo.ds")
$

define_application_code(:govenor_houses,:location,1,:point,:gas_netwo
rk)
$

define_rwo_table(!current_dsview!,:govenor,
      vec(vec(:govenor_id,:ds_uint)),
      1,1008,"rwo.ds")
$

define_application_code(:govenor,:location,1,:point,:gas_network)
$

define_rwo_table(!current_dsview!,:meter,
     vec(vec(:meter_id,:ds_uint)),
     1,1009,"rwo.ds")
$

define_application_code(:meter,:location,1,:point,:gas_network)
$

define_rwo_table(!current_dsview!,:surface_box,
     vec(vec(:surface_box_id,:ds_uint)),
     1,1010,"rwo.ds")
$

define_application_code(:surface_box,:location,1,:point,:gas_network)
$

!current_dsview!.commit()
$
```

GAS_EXEMPLAR.MAGIK

```
# define the exemplar for the pipes.

gis_ds_view.declare_record_exemplar(:pipe,:pipe, vec(),
   vec(:rwo_record_mixin,:rwo_sysid_mixin))
$
pipe.define_shared_constant(:visible_fields,vec(:inst_date,:diameter)
,_false)
$

# define the method for the pipes.

_method pipe.init_table()

 _self.declare_autogenerate (:|set_key_for()|)

_endmethod
$

# define the exemplar for the valves.

gis_ds_view.declare_record_exemplar(:valve,:valve,vec(),
  vec(:rwo_record_mixin,:rwo_sysid_mixin))
$
valve.define_shared_constant(:visible_fields,vec(:valve_id),_false)
$

# define the method for the valves.

_method valve.init_table()

 _self.declare_autogenerate (:|set_key_for()|)

_endmethod
$

# define the exemplar for the raster maps.

gis_ds_view.declare_record_exemplar(:raster_map,:raster_map,vec(),
  vec(:rwo_record_mixin))
$

raster_map.define_shared_constant(:visible_fields,vec(:sheet_ref),_fa
lse)
$

# declare the method for the raster maps.

_method raster_map.init_table()

 declare_mandatory(:raster_map,:sheet_ref)

_endmethod
$
```

MAIN.MAGIK

```
_method pipe.determine_style()


      _if _self.status = "n" _then
            _if _self.type = "mp"
                  _then _return 2
            _elif _self.type = "ip" _then
                  _return 3
            _elif _self.type = "sg" _then
                  _return 4
            _else _return 1
            _endif
      _endif

      _if _self.status = "ab" _then
            _if _self.type = "mp" _then
                  _return 6
            _else _return 5
            _endif
      _endif

      _if _self.status = "as" _then
            _if _self.type = "mp" _then
                  _return 8
            _else _return 7
            _endif
      _endif

      _return 9
_endmethod
```

VALVE.MAGIK

```
_method valve.determine_style()


      _if _self.status = "c"
            _then
            _return 2
      _elif _self.status = "o" _then
            _return 1
```

Appendix F

The V$SYSSTAT pseudo-table is used by ORACLE to store database statistics which may be displayed via various ORACLE utilities. The pseudo-table is constructed in memory when the database is initialised, and may be viewed directly by the user.

```
S#      NAME                               CLASS    VALUE
--      ------------------------------     -----    --------
0       cumulative logons                  1        307
1       current logons                     1        9
2       cumulative opened cursors          1        19198
3       current opened cursores            1        17
4       user commits                       1        11948
5       user rollbacks                     1        276
6       user calls                         1        319471
7       recursive calls                    1        300381
8       recursive cpu usage                1        0
9       session logical reads              1        1392435
10      session stored procedure space     1        0
11      CPU used when call started         128      0
12      CPU used by this session           1        0
13      session connect time               1        0
14      process last non-idle time         128      0
15      session memory                     1        1201973
16      max session memory                 1        9546989
17      messages sent                      128      26684
18      messages received                  128      26684
19      background timeouts                128      510287
20      session pga mode                   1        21449404
21      session max pga memory             1        21478436
22      enqueue timeouts                   4        13
23      enqueue waits                      4        24
24      enqueue deadlocks                  4        0
25      enqueue requests                   4        68501
26      enqueue conversations              8        2486
27      enqueue releases                   8        68472
28      db block gets                      8        146695
29      consistent gets                    8        1252109
30      physical reads                     8        195243
31      physical writes                    8        24202
32      write requests                     8        5853
33      summed dirty queue length          8        2079
34      db block changes                   8        95001
35      change write time                  8        0
36      consistent changes                 8        2120
37      write complete waits               8        635
38      write wait time                    8        0
39      buffer busy waits                  8        89
40      busy wait time                     8        0
41      redo synch writes                  8        11524
42      redo synch time                    8        0
43      DBWR exchange waits                8        0
44      exchange deadlock                  8        0
```

```
45      free buffer requested                8       203575
46      dirty buffers inspected              8       1568
47      free buffer inspected                8       3081
48      free buffer waits                    8       16
49      free wait time                       8       0
50      DBWR timeouts                        8       254416
51      DBWR make free requests              8       13071
52      DBWR free buffers found              8       82626
53      DBWR lru scans                       8       13182
54      DBWR summed scan depth               8       91466
55      DBWR buffers scanned                 8       88365
56      DBWR checkpoints                     8       296
57      calls to kcmgcs                      128     117246
58      calls to kcmgrs                      128     0
59      calls to kcmgas                      128     12234
60      redo entries                         2       54912
61      redo size                            2       17995651
62      redo entries linearized              2       0
63      redo buffer allocation retries       2       26
64      redo small copies                    2       54887
65      redo wastage                         2       4803610
66      redo writer latching time            2       0
67      redo writes                          2       13864
68      redo blocks written                  2       46075
69      redo write time                      2       0
70      redo log spoce requests              2       59
71      redo log space wait time             2       0
72      redo log switch interrupts           2       0
73      redo ordering marks                  2       0
74      background checkpoints started       8       22
75      background checkpoints completed     8       21
76      table scans (short tables)           64      6817
77      table scans (long tables)            64      5626
78      table scan rows gotten               64      4577154
79      table scan blocks gotten             64      398405
80      table fetch by rowid                 64      307729
81      table fetch continued row            64      88
82      cluster key scans                    64      11180
83      cluster key scan block gets          64      21331
84      parse time cpu                       64      0
85      parse time elapsed                   64      0
86      parse count                          64      367754
87      sorts (memory)                       64      287
88      sorts (disk)                         64      0
89      sorts (rows)                         64      448
90      cursor authentications               128     64175
```