National Library of Canada
Bibliothèque nationale du Canada

Canadian Theses Service
Service des thèses canadiennes

Ottawa, Canada
K1A 0N4

NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30, and subsequent amendments.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30, et ses amendements subséquents.

Canada

DESIGN AND IMPLEMENTATION OF A NATURAL LANGUAGE BASED EXPERT SYSTEM FOR A MULTIMEDIA DATABASE

by

GOUTAM K. SHAW

MAY 1990

A thesis

submitted to the School of Graduate Studies and Research

in partial fulfillment of the

requirements for the degree of

Master of Applied Sciences

in Electrical Engineering

OTTAWA-CARLETON INSTITUTE FOR ELECTRICAL ENGINEERING

Department of Electrical Engineering

Faculty of Engineering

University of Ottawa, Ontario, CANADA

© Goutam K. Shaw, Ottawa, Canada, 1990

National Library of Canada
Bibliothèque nationale du Canada

Canadian Theses Service
Service des thèses canadiennes

Ottawa, Canada
K1A 0N4

The author has granted an irrevocable non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of his/her thesis by any means and in any form or format, making this thesis available to interested persons.

The author retains ownership of the copyright in his/her thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without his/her permission.

L'auteur a accordé une licence irrévocable et non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de sa thèse de quelque manière et sous quelque forme que ce soit pour mettre des exemplaires de cette thèse à la disposition des personnes intéressées.

L'auteur conserve la propriété du droit d'auteur qui protège sa thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

ISBN 0-315-62338-1

Canada

UNIVERSITÉ D'OTTAWA
UNIVERSITY OF OTTAWA

I hereby declare that I am the sole author of this thesis.

I authorize the University of Ottawa to lend this thesis to other institutions or individuals for the purpose of scholarly research.

Goutam K. Shaw

I further authorize the University of Ottawa to reproduce this thesis by photocopying or by any other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

Goutam K. Shaw

CONTENTS

ABSTRACT

This thesis is concerned with the design and implementation of a Natural Language Based Expert System for a Multimedia Database for radiology applications. The design consists of three independent modules, namely the Natural Language Processor, the Expert System for query analyzing and result outputting, and the Database Management module (query execution module). These modules interact with each other through shared data and storage areas. The thesis emphasizes the integration of an expert system with a database management system to retrieve multimedia data stored in the database.

At this stage of the development, all the three modules can interact with each other. The input to the system is a query in English for retrieving the data from the database. The system can check the syntax and semantics of the input queries, generate as well as execute equivalent database queries, and produce adequate results. The Natural Language Interface and the query and result analyzers are written in MPROLOG on a SUN workstation, whereas the database software module is written in C with embedded SQL commands for retrieving the data managed by ORACLE Relational DBMS.

Examples of data modeling and the system organization are discussed and some sample runs are illustrated.

LIST OF FIGURES

LIST OF TERMS

AI	Artificial Intelligence
BNF	Backus-Naur Form
CSG	Context Sensitive Grammar
CFG	Context Free Grammar
DB	Database
DBMS	Database Management System
DBS	Database System
DCA	Data Communication Area
DDA	Data Descriptor Area
ES	Expert System
IE	Inference Engine
KB	Knowledgebase
KBS	Knowledge Based System
NL	Natural Language
NLP	Natural Language Processing
NLI	Natural Language Interface
RBS	Rule Based System
SQL	Structured Query Language
UFI	User Friendly Interface

To my Daughter Garima....

ACKNOWLEDGEMENTS

First, I would like to thank Prof. Nicolas D. Georganas, my thesis supervisor, for his direction, guidance and advice. His time and patience committed in communicating his knowledge, judgement, and experience in Multimedia Applications has been rewarded with the results of the research presented in this thesis. I would also like to thank him for introducing me to the new field of Expert Database Systems.

Next, I would like to thank Dr. Ahmed Karmouch, for his valuable advice and guidance from time to time in the area of Expert System - Database Integration.

Next, I would like to thank Mr. Dominique Vital, one of the research engineers developing the multimedia database applications, for helping me understand the prototype developed and letting me acquainted with the whole system at the University of Ottawa Medical Communications Research Center.

Next, I would like to thank my colleagues whom I shared the same office with, for always standing for me.

Next, I would like to thank the department of Electrical Eng. for providing good working conditions for computing and other administrative facilities.

Finally, I would like to thank my wife Geeta and my daughter Garima for their patience and understanding throughout the course of this thesis elaboration.

Chapter 1

Introduction

A broad set of user applications ranging from office automation to command and control involves the handling of information in a variety of representations. Often a single activity will involve handling several types of media such as digital imagery, voice, video, textual information etc. [Blumberg 89]. The information may be textual but have components with different formats such as forms, entries managed by DBMS, spreadsheets, or graphics. The effective handling of this kind of complex combination of data is becoming necessary. This is viable in the sense that, with the advent of powerful computers and new technologies, computer based user applications can now significantly enhance the natural perception of human beings. Textual data is not the only form of information which can now be handled by computers. Instead media such as graphics, video, voice etc. can become part of the computer presented data and present the user with more viable and powerful tools compared to their textual counterpart.

1.1 Multimedia Technology : An Introduction

If a computer is showing a graph, playing a tune, and displaying a text message, that is not a multimedia application [Robinson 89]. But if it plays music from a compact disk while animating the graph and displaying instructional messages then that is a multimedia application. Multimedia technology combines images, video, graphics, audio and text with standard data processing. Multimedia applications use the computer

i) to integrate and control diverse electronic media and

ii) to provide computational power for data manipulation in different media.

If data in different media are logically connected and the entire application is interactive then this is defined as a hypermedia application. Computer assisted multimedia document processing is a new technology for thinking, learning, and communication [Ambron 88]. Multimedia applications can be examined in three parts : media, technology, and products. Fig. 1.1 provides an illustration of this statement.

Typical media are text, audio and visuals. Multimedia applications are made possible by the technologies of optical storage and fast computing. Lastly, various multimedia products are just emerging. They are likely to touch all major aspects of computer assisted applications in our daily lives. Attempting to implement multimedia applications the problem which arises is to transform and organize this multimedia data in a manner acceptable to the computer, i.e. to represent the multimedia data with the help of existing data organization and management schemes. The database technology together with file management systems provides tools for storing, retrieving, and manipulating multimedia documents.

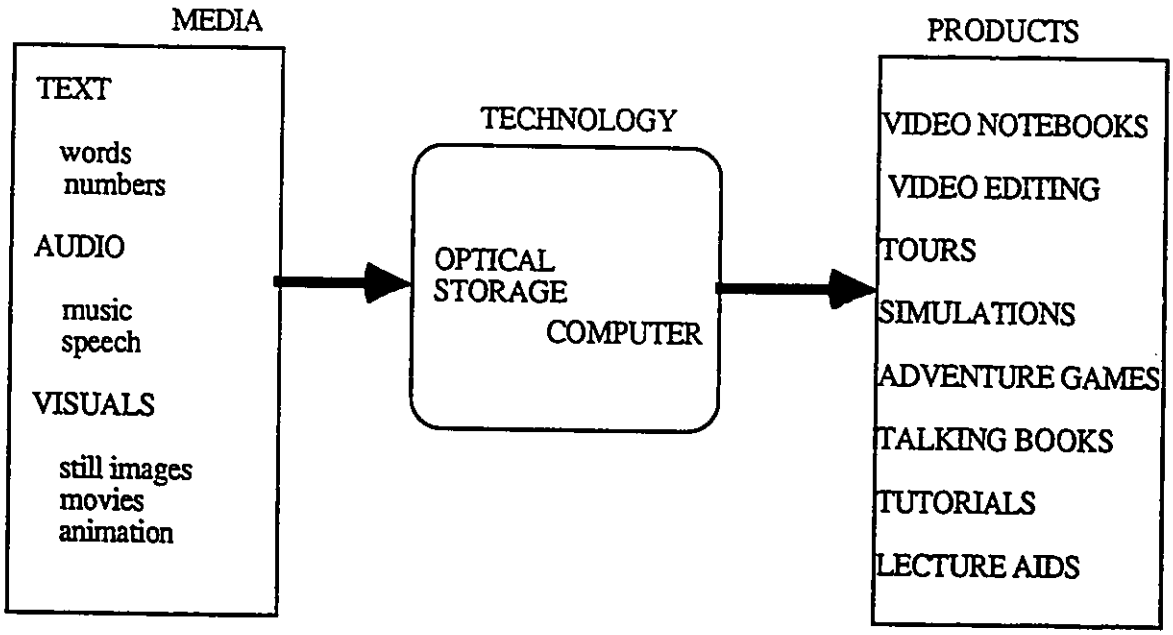


Fig. 1.1 Multimedia elements

1.2 Multimedia Databases

The multimedia applications are changing the conventional database technology. Multimedia databases have to handle more than character fields. They have to store digitized images, text documents, digitized voice, etc. and retrieve and manipulate them as well. As the relational data model has been proved viable for implementing databases, an extension of it can serve as a tool for implementing multimedia databases which are supposed to manage large, unstructured data objects. This extended relational model will provide facilities for storage, retrieval and manipulation of structured as well as unstructured data. The only difference between the conventional DBMS and multimedia DBMS is linking data objects of different media to their respective devices.

This kind of multimedia database uses a new concept of datatype called binary large objects (BLOB) [Shelter 90]. BLOB defines those fields in a record which contain the objects. Text and byte may be considered as BLOB datatypes. Text BLOBs may contain valid text characters (as defined by the attribute datatypes of a relational model), whereas byte BLOBs may contain any digitized data (voice, image, object code module etc.). Since the entire byte stream of BLOB datatypes may be very large and may not fit into the column width of a relation, there should be some mechanism to logically and physically relate the actual stored data with the data defined by the byte BLOB datatype in a relation. Two of such mechanisms can be as follows.

i) Link the objects with the help of individual physical pointers.

ii) Place the entire BLOB column of a relation on a separate partition of either the same or different secondary storage devices. A BLOBSPACE can be defined for denoting a logical region of the database that contains the columns of BLOBs. A BLOB column placed in the BLOBSPACE is linked to its corresponding entry in the record. This allows the user to perform high volume applications without large objects at optimal seeds.

Other database management problems, such as consistency, integrity, security remain as in conventional DBMS. In multimedia databases they can be resolved by the same methods, e.g., locking mechanisms for a better consistency control, and rule based systems for checking integrity and maintaining the security of the database. Multimedia databases require a slight modification of these procedures to suit their environment. Researchers are working for improving the overall multimedia DBMS performance by modifying data logging and recovery procedures (specially for unstructured data). The BLOBs can be retrieved and manipulated as easily as their text counterparts with the help of bit mapped devices.

All this seems to be good enough when the application is not large and does not involve a large number of BLOBs. The user can be confused by the application if it involves too many BLOBs, hence a huge database is required to work with. To make the life of an unskilled user simpler it has been proposed by scientists to create intelligent databases where the details of data are transparent to the user. This means coupling expert systems and databases in such an efficient way that it is user-friendly, provides good performance characteristics and does not require only skilled persons to be its sole users. There are many coupling techniques and tools proposed and one should choose one or define one's own according to one's design requirements. More on the topic of expert database systems (intelligent databases) will be discussed in the following chapter.

1.3 Thesis Research Areas

The work reported in this thesis constitutes a beginning effort at the University of Ottawa Medical Communications Research Center for designing and implementing an user friendly expert database system for radiology data communications. The Center has developed a Multimedia Medical Communication System currently dedicated to radiology communications [Karmouch 90]. This system provides radiologists and physicians with a set of workstations from which radiology information can be captured, processed, updated, retrieved and consulted. The radiology information involves different media. Textual form is used for storing

the text documents pertaining to the personal information of the patients. X-Rays and CT scans are stored in digitized form. Voice annotations of the radiologists referring to patients' examinations are stored along with the images.

The involvement of data from different media requires a management system which would provide coherent structures to the data of different media. A database management system solves this problem and as discussed above a relational data model is able to provide this facility.

Fig. 1.2 illustrates the organization of the multimedia database configuration. The workstations interact with the database server by sending queries to the database server and receiving the results. The server and the workstations are connected via Ethernet. Each workstation is built around the Compaq 386/20 microcomputer, equipped with a 60 megabyte system disk and 13 megabytes of RAM. An EGA compatible display controller and monitor serves as the workstation control screen. The image screen, a high resolution monochrome monitor, provides an 8 bit pixel image of resolution 1280 by 1024 with a 2 bit graphics overlay plane. The Image Entry workstation is also equipped with two devices for digitizing films and documents :

i) A laser film scanner is capable of digitizing all standard film sizes upto 14" by 17". The maximum image resolution is 2000 by 2400 for a 14" by 17" film. All films are digitized to 1024 grey levels.

ii) A Scan Jet paper document scanner provides the capability of digitizing pages upto 8.5" by 11" at a resolution of 100 dots per inch.

The image is stored in two levels of resolution, i.e., 1000 by 1000 and 2000 by 2000. Due to limited network bandwidth, the size of image files, and the requirement for rapid display, a reduced resolution copy of all images is normally made available as soon as an image is digitized to the local image store at each workstation. The high resolution image is retrieved from the remote server for a better view and is termed as "zoom". In addition to the above

equipment , each workstation is equipped with a handsfree telephone. The voice terminal provides a link to the Voice Mail system for dictation and playback of voice reports.

The multimedia server consists of a SUN-3/160 workstation and a Northern Telecom Meridian Mail/SP Voice Mail system. The SUN workstation runs the database system in the standard client-server mode. The users workstations send requests to the database server, which processes them and returns back the result. The radiology data are logically divided into the following two categories.

i) The patient folder - It contains all the information available on a patient.

ii) The examination report - It consists of different examination reports of a patient, e.g., X-Ray images, voice annotations to the reports, textual comments etc.

The patient folder stores information such as name, address, date of birth, creation date, etc. whereas the examination report is divided into several logical parts equal to the number of reports of a patient. This partitioning is conceived from the fact that each patient has several reports and each examination report may consist of more than one type of examination.

The data model of this multimedia database ensures the consistency and security of the data of different media [Vital89]. The work reported herein has been done for providing the user an intelligent access to the multimedia database through the client workstations.

The volume of data involved in the multimedia application makes us search for an integrated way of handling the multimedia data which could be operated with ease even by a non-technical user. This gave birth to the problem of designing an intelligent man-machine interface for the multimedia radiology application which is a result of `integration of expert system and multimedia database. The user interface has been chosen to be based on a natural language query system and the intelligence is incorporated into the system by virtue of an expert system. The expert system in turn is interfaced to the actual multimedia database

management system which helps in retrieving, storing and manipulating the multimedia data. The choice of natural language interface and the expert system approach for solving the problem have been justified in the following chapters. The thesis covers the following major topics :

* Definition and understanding of expert systems, databases and their integration.

* Definition and understanding of adaptive user interfaces, in particular natural language processing and its front-end applications.

* Design of a natural language based expert database system for accessing a radiological multimedia database.

* System implementation on a SUN workstation using MPROLOG, C, and embedded SQL and/or SQLPLUS interpreter.

The thesis is organized as follows. Following this introductory part, chapter 2 presents an understanding and literature survey of database and expert systems and the ways of their integration. It also contains a discussion on research areas in adaptive user interfaces, specially natural language interfaces and their pros and cons. Chapter 3 discusses the issues of designing a natural language based expert system for multimedia database for a radiology application. It also discusses the various possibilities of implementation and other implementation issues. Chapter 4 discusses briefly some enhancement issues pertaining to the implementation. Chapter 5 provides with some sample runs, results and comments about the performance of the expert database system module. Conclusion and suggestions for future work are reported in chapter 6. An appendix contains the program modules.

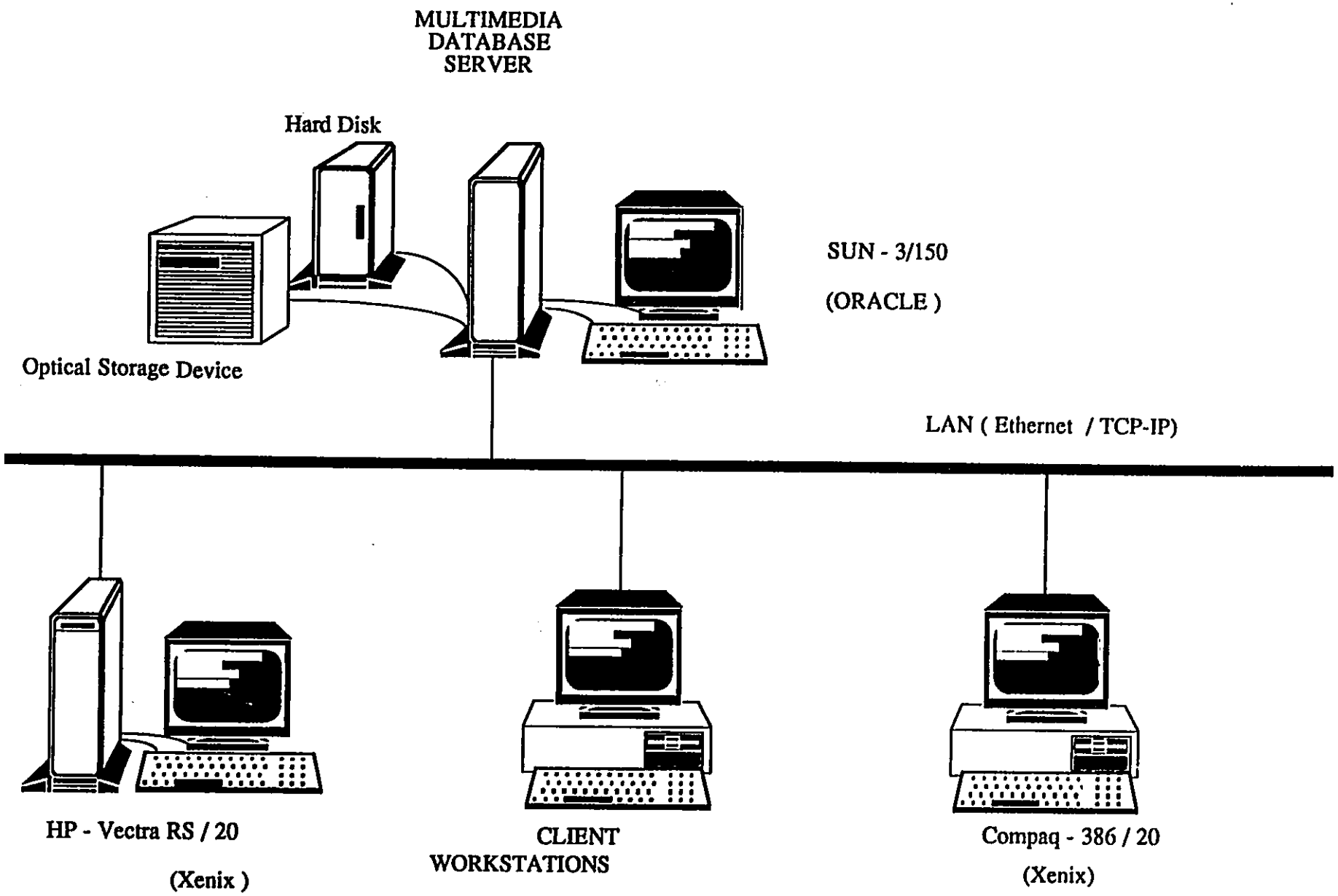


Fig. 1.2: Multimedia database configuration

Chapter 2

Expert Database Systems : A Perspective

If we look around us we find that we are engulfed by a world of data. Data means any kind of information which is useful to determine some or the other fact. Go anywhere and you land up either providing information or acquiring information. It is an age of information explosion. Information has become so vital that it is representing every aspect of life, be it a bank, or a business organization, or an academic institution. The management of information (data) by the computers is known as database management. A database is nothing but a collection of data which on the whole, represents the facts about a subset of real world objects or entities. Since the information is money, the faster the best information is accessed, the better [NewquistIII88]. This gave birth to the database management system (DBMS) software, which is responsible for efficient storage, retrieval, update, and manipulation of data stored in computers. But with the use of DBMS software the world slowly started finding out that they are very inflexible. The inflexibility is revealed in the sense that they are only good enough for retrieving the data stored, but do not have decision making capabilities. For example, one can retrieve the salaries of the employees working in the organization with the help of DBMS, but can not analyze the retrieved data for further decision making process with the help of same DBMS. For that, one has to take help of some decision making software which takes the data outputted by DBMS as its input and produces its result. The other way round is to analyze the data manually. This limitation imposed a great concern over the viability of DBMS software in the world of intelligent decision making processes supervised by the computer [Schur 88]. But then came the solution to integrate DBMS with software called Expert System, capable of decision making. Expert system is a tool helping users to make decisions based on combination of certain rules embedded in it and the data stored in its knowledgebase and/or provided to it by external means.

The advent of expert database systems started revolutionizing the world of computer assisted data retrieval and analysis. In order to understand the functionality of the expert databases, one requires a fair idea of its constituent parts which are as follows.

i) The data management part which has powerful access mechanisms for retrieving and manipulating the data stored in secondary storage, as found in conventional DBMS.

ii) The reasoning part which is capable of analyzing the data and taking decisions based on the current analysis of data available to it. This deductive nature of reasoning is found in expert systems.

The later sections of this chapter discuss the database concepts, expert systems and focus on some database - expert system integration strategies together with their feasibility analysis and methods of integration. The last section is devoted to one of the major issues in any system design - the user interface, particularly the natural language interface and its pros and cons.

2.1 The Database Concepts

The birth of database technology was due to increase in the volume of data to be handled. The technology provided a better management of data in terms of access strategies, retrieval, update, deletion etc. The DBMSs are categorized into following three classes depending on the corresponding data models.

i) Hierarchical data model

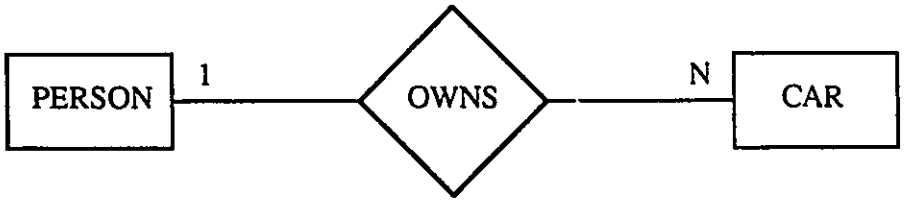
ii) Network data model

iii) Relational data model

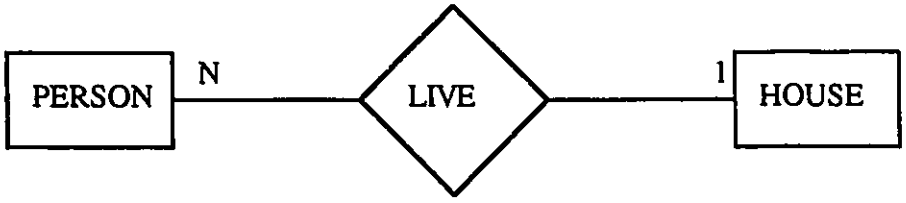
Before discussing the data models themselves, it would be appropriate at this stage to first understand the root concept which helps to understand all data model concepts in a better and organized way. This is known as entity - relationship model of data. The entity - relationship model will also be discussed further as a tool for semantic modeling of data.

2.1.1 The Entity - Relationship Model

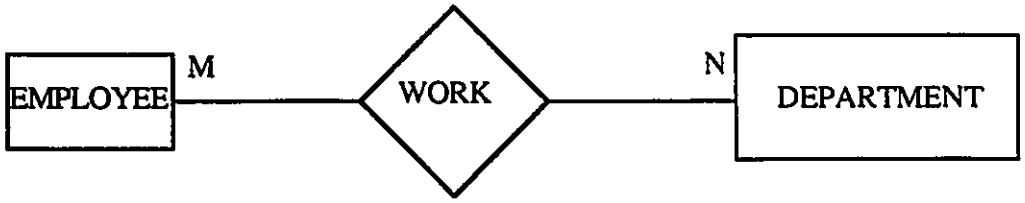
The entity - relationship model [Chen 76] is a tool for representing a relationship between two or more entities. The entities could be any of real world objects, records etc. The objects are related to each other by means of a relationship name and the relationship also includes the cardinality of the relation, i.e., also shows how they are related through that relationship. Let us take for example, two objects PERSON and CAR. They are related via a



(a) One to many relationship



(b) Many to one relationship



(c) Many to many relationship

Fig. 2.1 Entity - relationship diagram

relationship called OWNS. Now to represent the cardinality of the relationship we can say that one person can own more than one car. This represents a 1:N (one to many) relationship between objects PERSON and CAR and can be illustrated diagrammatically as shown in fig. 2.1 (a). Similarly, fig. 2.1 (b) represents a N:1 (many to one) relationship LIVE between entities PERSON and HOUSE, meaning more than one person live in one house. Consequently, fig. 2.1 (c) depicts the M:N (many to many) relationship WORK between EMPLOYEE and DEPARTMENT. In simple words it means that one employee can work in more than one department as well as one department can have more than one employee.

The entity - relationship models are used as first step of defining any database design application irrespective of the data model used for the actual database design.

2.1.2 The Hierarchical Model

The hierarchical data model was one of the proposals which the first commercially available database systems were based on. A hierarchical database consists of an ordered set of trees - more precisely, an ordered set consisting of multiple occurrences of a single type of tree. A tree type includes a single "root" record type and an ordered set of zero or more dependent subtrees. A subtree type again in turn includes a single record type (root of the subtree) and an ordered set of its dependent subtrees, and so on. The entire tree consists of a hierarchic arrangement of record types [Date 86, Schur 88] . Fig. 2.2 shows the conceptual structure of data hierarchy of the hierarchic database model. The model represents the one to many relationship between objects.

As an example, we can consider a database shown in fig. 2.3 which represents the internal education system of an industrial company. The education department of the company runs a number of training courses for the company employees. Each course is offered at different locations and has some teachers associated with it. From the hierarchic structure of fig. 2.3 following information can be extracted :

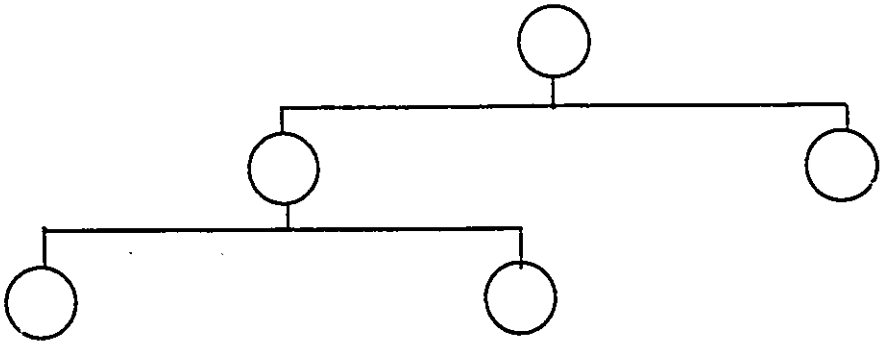


Fig. 2.2 Hierarchical structure of data

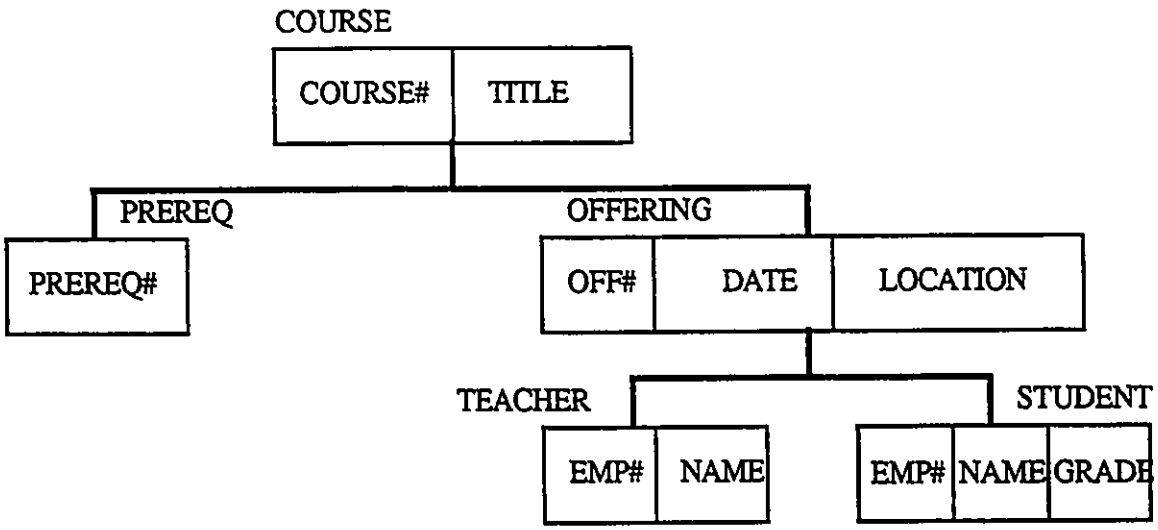


Fig. 2.3 Education department database

– COURSE record type containing field types course# and title, has two dependent record types, e.g., PREREQ and OFFERING. It means for each course# there exists a number of prerequisite courses (each identified by an unique prereq#) and a number of offerings (each identified by an unique off#).

- For each offering of a given course there exists a number of teachers associated with that particular course (uniquely identified by emp#) and a number of students attending the course (uniquely identified by emp#).

- A distinct off# determines unique date and location for record type OFFERING.

- A distinct emp# determines the name field in TEACHER and name and grade fields in STUDENT.

COURSE record type is known as parent record type of its child record types PREREQ and OFFERING. Similarly, OFFERING is parent for its child TEACHER and STUDENT. The database contains five record types. A sample tree of the database in fig. 2.4 is helpful in understanding the hierarchy.

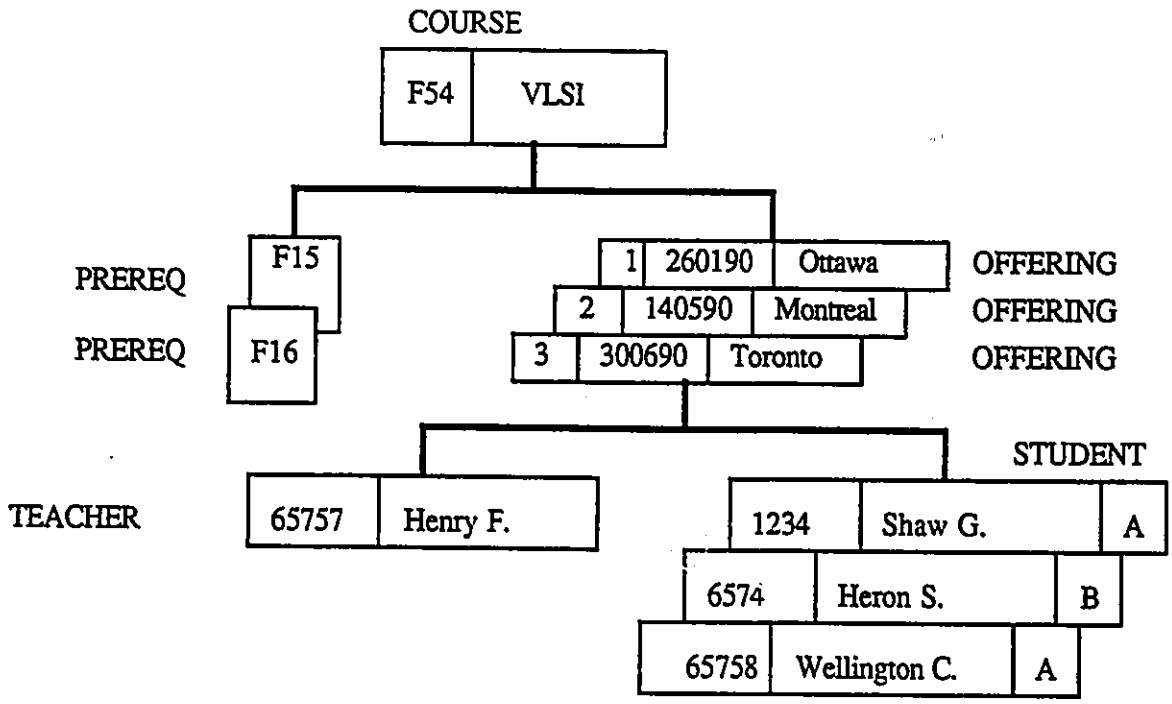


Fig. 2.4 Sample tree of the database

2.1.3 The Network Model

The network data structure is considered to be an extension of the hierarchic data model. The main distinction between these two models is that, as opposed to the hierarchic model where a child record type can have only one parent record type, the network model may include child records having more than one parent record type [Date 86]. The data model structure can conceptually be represented as shown in fig. 2.5. The network model can be visualized as entity - relationship model having many to many relationship.

The Network data model consists of two sets namely, a set of records and a set of links. A link is established between two record types having a 1:N relationship among themselves, where one is the owner and the other is the member (like the hierarchic model). The link is known as set type. Each occurrence of owner record type is the parent of exactly one occurrence of link (set type). And each occurrence of member record type is a child in at least one occurrence of link. For example, the network version of the education database is illustrated in fig. 2.6.

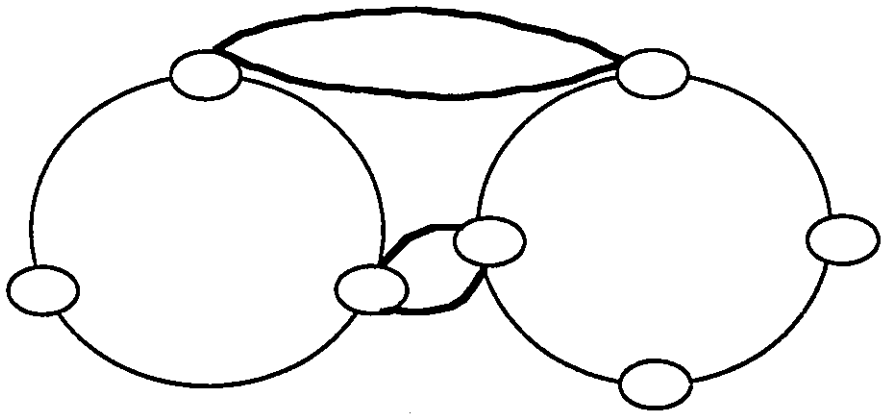


Fig. 2.5 The network data model structure

It contains four record types, i.e., COURSE, PREREQ, OFFERING, and EMP. The links or set types define the 1:N relationship between the record types involved. Letters "1" and "N" on the record types denote owner and member correspondingly. Fig. 2.7 shows a sample extract of the same database for the network version of the data model.

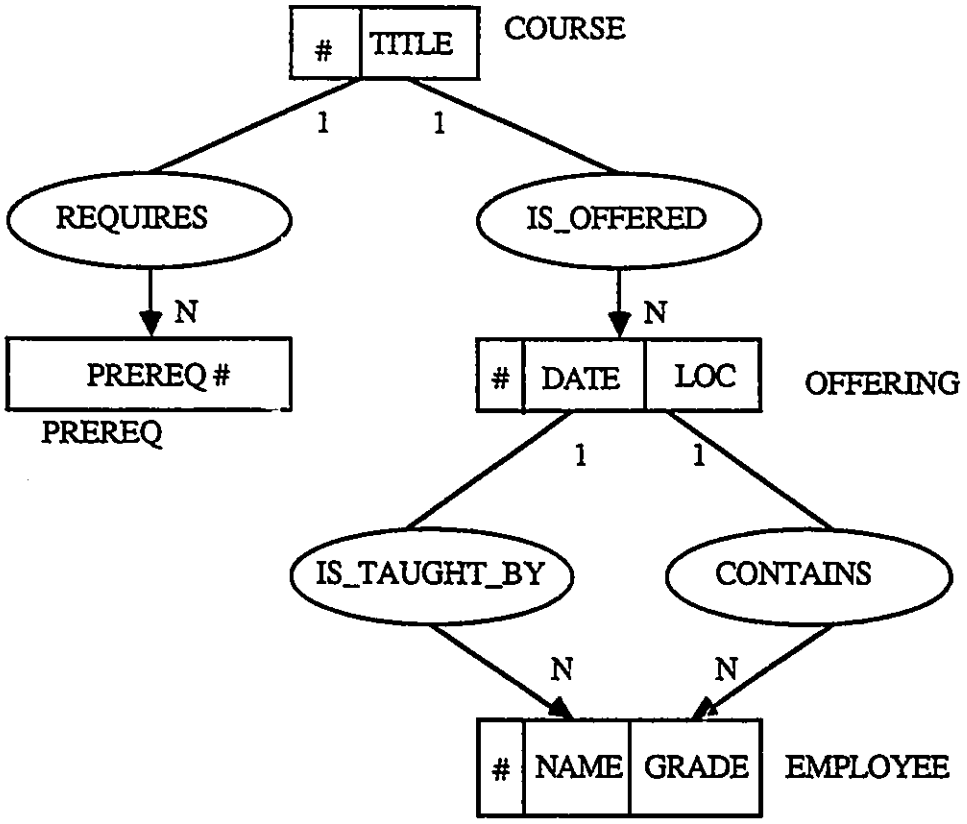


Fig. 2.6 Network version of education database

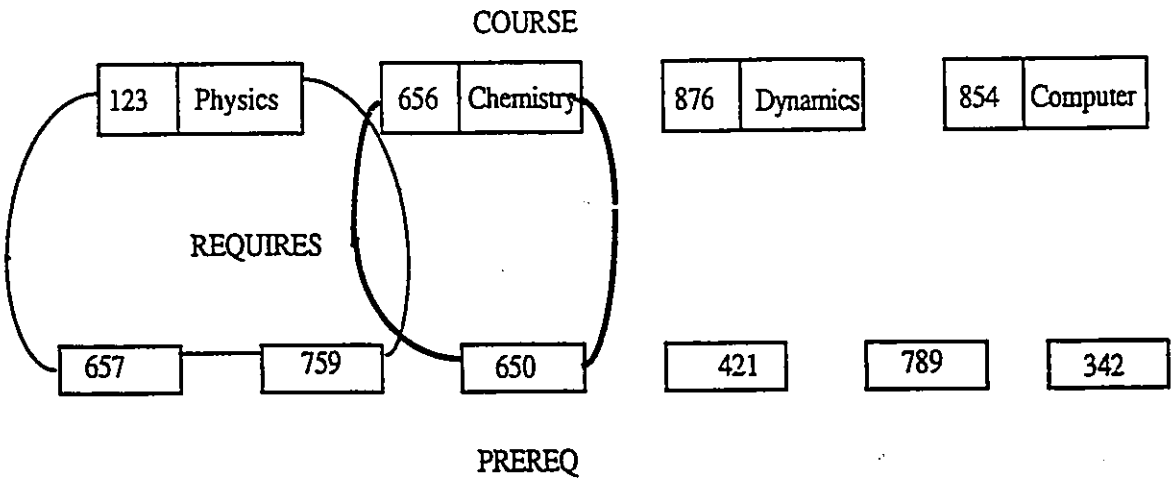


Fig. 2.7 A sample of network database organization

2.1.4 The Relational Model

The relational model data structure is the latest of all. It is considered to be a powerful model backed by mathematical theories. The database built upon this model is known as relational database. The relational model of representing a data structure is perceived by the users as nothing but a collection of tables [Date 86]. It can be conceptually viewed as shown in fig. 2.8. In a relational database a table is known as a relation and is given an unique name. The relation consists of several columns which show the cardinality of the relation. Each column is represented by a fixed data structure called attribute. Each row of the relation is termed as a tuple and contains the values of the attributes in their corresponding columns. Each relation contains a primary key (a set of one or more attributes which defines the rest of the attributes in the relation). It suggests that there can be found no two tuples in the relation having the same key value [Yang 86]. The relational data model consists of three parts which are as follows.

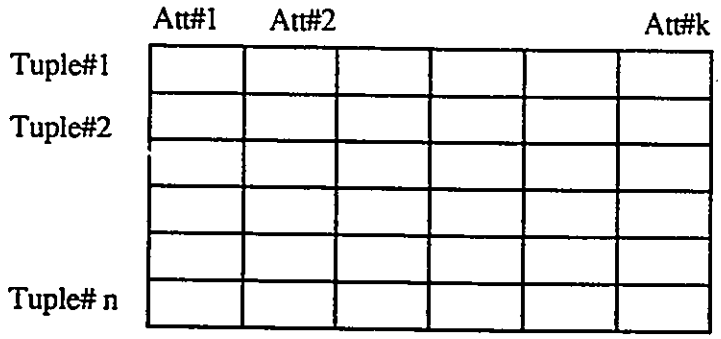


Fig. 2.8 The concept of relational model

i) Structural part : It includes the n-ary relations (attributes and tables) and domains. Values of each of the attributes are defined by their underlying domain.

ii) Integrity part : It consists of two integrity rules, i.e., entity integrity and referential integrity.

- The entity integrity rule tells that primary key values must not be null.
- The referential integrity rule deals with the foreign keys and suggests that if a relation contains a foreign key (other than the primary key) which matches a primary key in some other relation then every value of the foreign key in the first relation should be equal to the value of the primary key in some tuple of the second relation.

iii) Data manipulation part : This part provides a set of algebraic operators for data manipulation which are stored in the relations of the database.These operators have been developed on the basis of relational algebra (or relational calculus). Some of the relational algebra operators are select, project, join, divide, union, intersection, difference, product etc. These operators not only facilitate better data retrieval strategies but also provide the tool to construct different relations and thus are helpful in database design.

The relational model provides a better and unified approach towards building databases. The main reason of network and hierarchic approaches for not being popular is that they lack the mathematical support. In the words of C.J.Date - The *system is not good just because it supports the relational model, on the other hand, if it does not support that model, then it very likely is not good.*

Let us take an example and try to understand some of the basic algebraic operators like select, project, and join. All other operators can be defined in terms of these operators. Fig. 2.9 shows two relations STUDENT and COURSE containing some attribute values in their tuples. STUDENT# is the primary key for the relation STUDENT whereas COURSE# and STUDENT# are the primary and foreign keys respectively, for the relation COURSE. The relation shown in fig. 2.10 (a) is the result of a select operation on relation STUDENT for the student number 12345. The select operation produces all the tuples which match the selection criterion. In this case the selection criterion is the value of the student number. Fig. 2.10 (b) contains a relation which is the result of a project operation on the relation COURSE. The query was to project the student numbers who have 'A' grade. Finally, fig. 2.10 (c) contains a relation which is produced by a join operation involving both the relations. The join is performed on a common key (in this case it is STUDENT#). The effective relation contains all the attributes of both the relations. The procedure of join is that each tuple of one relation is compared with all the tuples of another relation and whenever the values of the common key match, both the tuples are entered into the resultant relation forming a single tuple.

STUDENT

STUDENT#	NAME	TEL_NO.
12345	Shaw G.	564-4062
54321	Fox H.	819-3045

COURSE

COURSE#	STUDENT#	GRADE
ELG123	12345	A
ELG123	54321	B
ELG321	12345	A

Fig. 2.9 A sample student database

12345	Shaw G.	564-4062

Selection criterion STUDENT#=12345

(a) Select operation on STUDENT

12345
54321

Selection criterion "GRADE=A"

(b) Project operation on COURSE

12345	Shaw G.	564-4062	ELG123	A
12345	Shaw G.	564-4062	ELG321	A
54321	Fox H.	819-3045	ELG123	B

(c) Join operation on the common key STUDENT#

Fig. 2.10 Examples of basic relational operations

The relational data model is becoming standard for designing databases whereas network and hierarchic models are no more commercially viable. As long as the application is limited to data retrieval and updating, the relational model provides a powerful mechanism for data access and manipulation. But it lacks the power of reasoning. With the growth in development of artificial intelligence products specially in knowledge - based expert systems, the significance of data syntax started giving way to the data semantics. In the world of reasoning, the raw data is of no more interest, instead more interest is drawn to what it means -

the semantics. The expert systems (or the rule based systems) put much stress on the semantics of data whereas DBMS is oriented towards the syntax of data. Some efforts have been done to increase the inferencing capability of conventional DBMS but still they lack the power of conventional inferencing mechanism of expert systems. This will be discussed later in this chapter.

2.2 The Expert System

As it has been mentioned briefly in the previous section, with the growing efforts in development of intelligent systems the researchers are concentrating more on devising machines which could simulate (fully or partially) the reasoning method of human beings. This culminated into a research trend called artificial intelligence. AI is the part of computer science concerned with designing intelligent computer systems, which exhibit the characteristics of human intelligence, i.e., learning, reasoning, solving problems, understanding language and so on [Barr 81]. One of the branches of the AI area is known as expert systems. Most often they are called rule-based or knowledge-based systems. The basic concept is - "*If* something happens *then* take some action". This rule, upto a certain extent, represents our way of thinking. Basically, an expert system consists of a bunch of rules which decide the action on the data available to them externally or internally (in its own knowledgebase). Fig. 2.11 illustrates a typical block diagram of an expert system. The inference engine provides the reasoning power and the knowledgebase provides the data to be operated on.

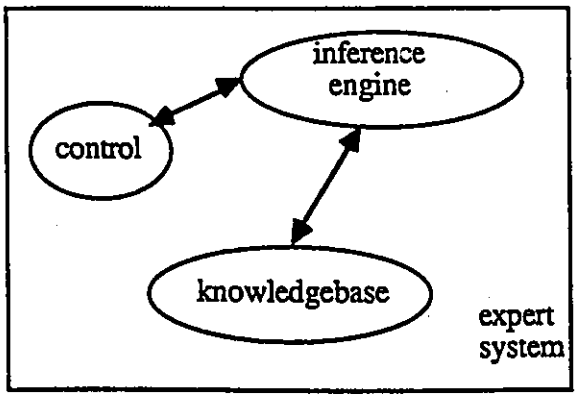


Fig. 2.11 Constituents of an expert system module

2.2.1 Components of an Expert System

The expert systems can be categorized on the basis of two guiding factors - first, the searching technique and second, the knowledge representation [Barr 81]. Let us try to understand these factors in some more detail.

2.2.1.1 Searching Techniques

The basic idea of searching techniques being used in AI is always to find a solution to the problem be it theorem proving, puzzle solving or winning a game. The search mechanism includes three major components which are as follows.

i) A database - which describes both the current task domain and the goal. As an example for information retrieval systems, the current situation is represented by a set of facts and the goal is the query to be answered.

ii) A set of operators - which consists of a a set of rules or modules of rules in an hierarchical structure. Execution of each rule of inference gives birth to new assertions in addition to the existing ones.

iii) A control strategy - which determines what operator to apply and when and where to apply it. Sometimes control is diffused in operators themselves. The control strategy affects the contents and organization of the knowledgebase. Each application of an operator changes the entire state of the search space. The control strategy investigates the operator execution sequence and also tries to undo a particular state if it does not yield the desired goal. This is known as backtracking. This governs the two types of reasoning mechanisms, namely, forward and backward reasoning.

A. Forward reasoning : This mechanism implies a control strategy which starts always from the initial state of the search space and keeps on trying to execute the operators until the goal (or the final state) is reached or no more alternative is left. This is also known as bottom-up search.

B. Backward reasoning : The backward reasoning mechanism unlike the forward reasoning technique, starts applying the operators to the goal and subsequently breaks the goal into several subgoals (as dictated by the operator). Then each of the subgoals are tried individually to be solved. Each of the subgoals may in turn have its own set of subgoals. This continues exhaustively until the operator representing the goal is satisfied. It is also named as top-down search. The backward reasoning technique easily supports the first order predicate calculus and horn clauses.

Most of the AI programs are based on the backward reasoning strategy. But still quite a few are available which apply both of them. This technique is called means-end analysis. The programs based on this consist of control strategy which after each execution of an operator calculates the difference between the current state and goal state and thus based on the result, determines which of the reasoning techniques to apply further in order to achieve the goal faster. The reasoning always requires a search method to invoke appropriate operators at particular situations. This is provided by a searching method buried in the program. Some of them are discussed below in brief.

The problem state space can be defined by a directed graph which consists of a set of one or more initial states, a set of operators for each state, and a goal state. A solution is found going from initial state to the goal state and the arc represents the transformation of one state into another by applying a set of operators. The traversing on the state space graph can be done in many ways which are as follows.

A. Breadth-first search : This method measures the distance of a sequence of operator, representing a node, from the initial state (node). If the number of arcs traversed till now is "n" then it considers other operator sequences at the same level "n" before considering operator sequences at the level "n+1". This search method ensures the shortest possible solution, if any exists.

B. Depth-first search : In this method the search follows a single path through the graph from the start node and terminates only if it reaches a goal node or a node which has no successor. In this case it may consider other backtracking algorithms. One of them is to try other alternatives from the previous node (its parent node).

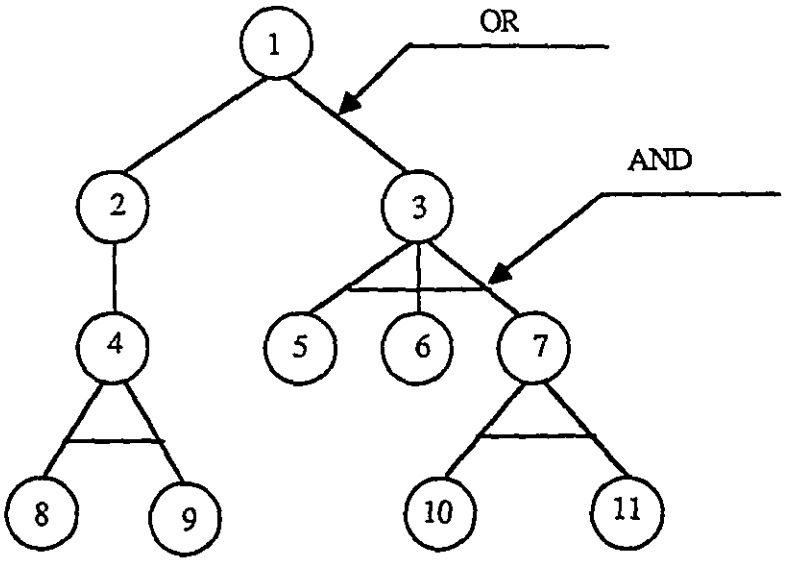


Fig. 2.12 An AND / OR graph

C. AND/OR graph search : This method of searching is based on logical connecting among the subgoals of a particular node. This means a rule representing a node may have several subgoals to be solved and these subgoals may be connected by AND or OR logic. In simple words each node containing an operator sequence is connected to its successor nodes via OR arcs. Each successor node may contain sequences which are ANDed together meaning that in order to solve that node all its ANDed nodes are to be solved together. Whereas OR represents an alternate path. Fig. 2.12 illustrates an AND/OR graph. The nodes ANDed (to be solved simultaneously) are joined by horizontal bars. Either of the breadth-first or depth-first searches can be applied to AND/OR graph to obtain a solution to a problem.

D. Other heuristic search methods : There are some heuristic search methods which when applied to state-space graphs yield faster solution algorithms. These algorithms imply some criteria which help in converging on the solution fast. A* optimal search, bidirectional search,Minmax procedure, Alpha-beta pruning are some of its examples. Their discussions are beyond the scope of this thesis. [Barr 81, Tanimoto 86] deal very deeply in this.

2.2.1.2 Knowledge Representation

The AI program supposed to simulate human intelligence requires some knowledge to be presented to it in any manner suitable to it. In AI the representation of knowledge is considered to be a combination of data structures and interpretive procedures, which helps the program to behave knowledgeably. There are several knowledge representation techniques available which can manipulate the data structures to make intelligent inferences.Some of the representation techniques are mentioned below.

A. Logic based representation : Classical way of knowledge representation is logic which evolved as a consequence of propositional and predicate calculi. The formal notion of the logic is either something true or false in a closed world system. Knowledge can be expressed by sentences like

All human beings have legs

which is known as proposition or formal logic. A proposition can be translated into a mathematical formula as shown below and is known as predicate calculus expression.

$\forall x.$ human_being(x) --> has_legs(x)

This reads, "for any object x in the world, if x is a human being then x has legs". This rule of inference is able to derive one fact if some other fact is known to be true. Now at this moment, if we add another expression (inference rule) to our system like

$\forall x.$ man(x) --> human_being(x)

which reads "for any x, if x is a man, then x is a human being", then the following can be inferred.

$\forall x.$ man(x) --> has_legs(x)

It reads " for any x, if x is a man, then x has legs".

The predicate calculus is an extended notion of propositional calculus and provides a means of a formal mathematical representation for the propositional logic. In the above expressions man(x),human_being(x), has_legs(x) are called predicates, whereas x represents the variable associated with the predicates. The predicates can be connected by connectives such as $\wedge$(AND), $\vee$(OR),$\neg$ (negation) and -->(implication). A predicate is either true or false.

Predicates can have more than one argument. There are two quantifiers used for predicate calculus expressions, i.e., ∀(for all) and ∃ (there exists).

Since predicate calculus is very general, two other notions have been added to this concept. First, the notion of functions which is unlike predicate not evaluated as to be true or false, but returns the value of the objects related to their arguments.For example, father_of(father_of(Bill)) will return the name of Bill's grandfather. The second notion is that of predicate equality. Two individuals are said to be equal if and only if they are indistinguishable under all predicates and functions. After addition of these notions to the predicate calculus it represents a variety of first order logic. A first order logic permits quantification over individuals but not over predicates and functions. It is both sound and complete. Logic based representation has an advantage that deductions are guaranteed correct to an extent that other techniques have not achieved. This is the reason of its popularity.

B. Procedural Representation : This representation evolved as a result of encoding control of the theorem proving process within a logic based system. In this kind of knowledge representation, the knowledge is contained in small programs called procedures and the control mechanism keeps track of their execution. The PLANNER programming language was built on this concept.

C. Production systems : The knowledge representation in the form of production systems was developed by Newell and Simon for creating models of human cognition [Newell 72]. It is a modular knowledge representation scheme. The knowledge is described by rules called productions. A production consists of a condition-action pair, i.e., IF <condition> THEN take <action>. The condition phrase may be a conjunctive/disjunctive normal forms of some subexpressions. One of the advantages in this type of representation is that the interaction between rules is minimized. The conditions in which rule is applicable are made explicit. The production systems bear similarity to the BNF (Backus-Naur Form) representation used in parsers. Many AI programs like DENDRAL, PROSPECTOR, MYCIN have been developed using a production system.

D. Frame representation : The most recently developed knowledge representation scheme is the frame. A frame is a data structure which contains procedural and declarative information in predefined relations. A real world object is defined by its generic structure and its instantiations. For example, fig. 2.13 illustrates a generic frame for a CAT and one of its instantiations. The generic frame defines the attributes (their names and number) in all the instantiations. Here attributes are, Self, Breed, Owner , Name. The attributes Owner and Name can be some attached procedures for finding out the owner and name from other frames. The idea of frame based knowledge representation comes from the relation data model used in DBMS.

Generic CAT Frame :

Self : an ANIMAL

Breed :

Owner : a PERSON

Name : a PROPER_NAME

THE_CAT Frame

Self : a CAT

Breed : Siamese

Owner : Mark

Name : Kiskis

Fig. 2.13 Frame based knowledge representation

E. Semantic Nets : One of the many ways of knowledge representation is in the form of semantic nets developed by M.R.Quillan [Minsky 68]. It is a psychological model of human associative memory. A net includes nodes which represent objects (events) and links between the nodes representing their relationship. Like the example given in fig. 2.14 illustrates the

concept "a MAN has LEGS" and "WILLIAM is a subset of MAN". This association automatically signifies that if an object has some property then a subset of that object also acquires that property, i.e., William has legs. This can easily be converted into logic representation. But the inferences drawn in a semantic net are not always valid unlike in the case of logic based representation. The program which manipulates the information extracted from semantic net, is responsible for deriving the valid inferences.

The searching and knowledge representation schemes presented above are the key features of an AI program. But the implementation of one or the other scheme totally depends on the nature of application and required efficiency.

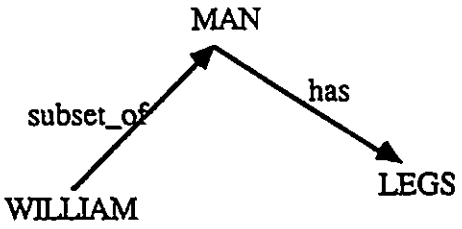


Fig. 2.14 A semantic net

2.3 Merging Expert Systems and Databases : A New Generation Called Expert Database Systems

Database and expert systems are the concerns of two separate fields of interest. DBMS lacks the properties of ES and ES in turn can not beat the data accessing capability of DBMS. A DBMS can not be rebuilt to qualify for an intelligent database systems due to its structural limitation. DBMS is oriented towards syntax and ES operates on the semantics, which are two opposite things. Another aspect is that the relational data model used in relational databases, has been accepted as a standard and the relational algebra, underlying this model, can be subjected to rigorous rule-based analysis which is a part of ES operations [Schur 88]. Fig. 2.15 - an example of an organization - shows how an expert system-database integration can smoothen the management of an organization. The ES provides certain business rules and policies to the office workers. These rules and policies are governed by the manager who in turn takes the data provided by the workers and modifies or makes policies. This is a closed loop electronic management environment where the manager decides the policy based on the data stored by the workers in the database and the workers are guided by the policies stored by the manager in the knowledgebase of the ES. This example shows one of the applications of the DBMS-ES integration.

The main distinction between an expert or intelligent database system and an ordinary database system is the inference engine and the rulebase (properties of ES) incorporated in it. The inference engine is a procedure to make logical inferences from a knowledgebase containing declaratively represented rules [Cohen 89]. The knowledgebase can be represented in the forms ranging from simple list of rules to an object-oriented hierarchy of nodes or semantic networks. Before discussing the techniques of integration, let us explore the similarity between database and knowledgebase systems.

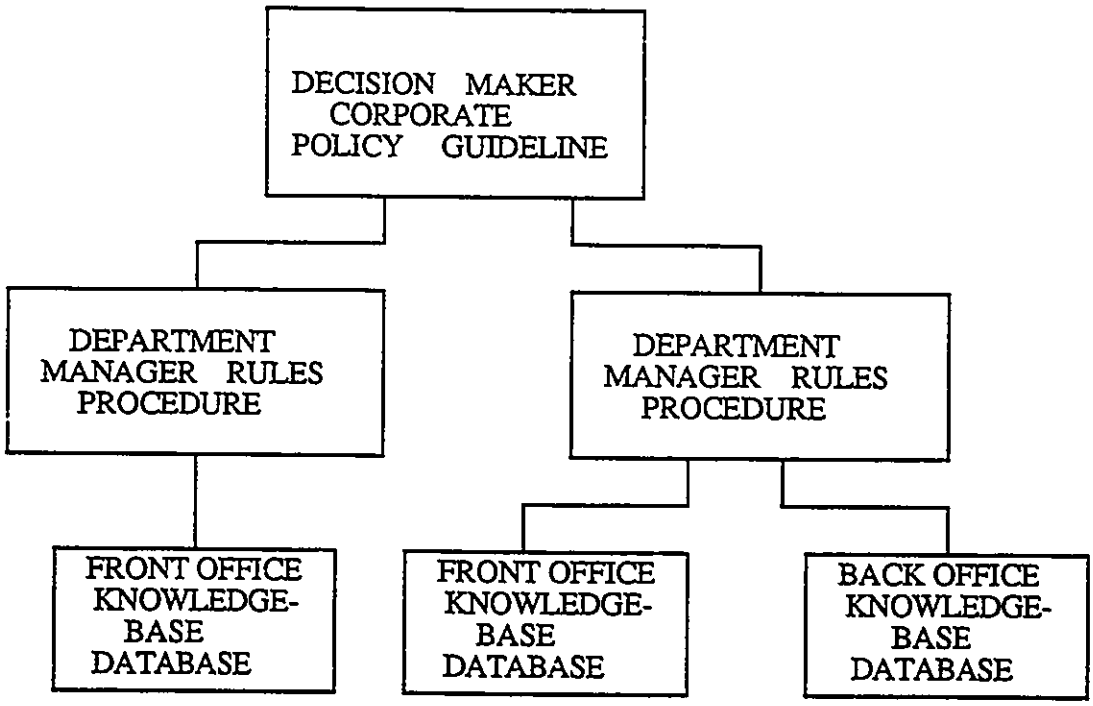


Fig. 2.15 Hierarchical model of corporate management

2.3.1. Similarity Between Database and Knowledgebase

Let us compare the relational model with the ES components. Actually the similarity discussed later, is used as the main tool to merge expert systems with relational databases. Relational databases have components like entity type, entity, attribute, and attribute value, whereas a knowledgebase consists of components like class, object, property, and property value. Each of the components of a database can be brought to a correspondence to the components of a knowledgebase as shown in table 1. For example in the database world of SUPPLIER, the columns of the table represent attributes of the entity SUPPLIER, i.e., SUPPIER#, SUPPLIER_ NAME, ADDRESS, etc. Each row of the table (relation) is unique and is known as instances of the entity. Similarly, if this has to be represented in the ES world, an object belonging to the class of suppliers has properties like, O_SUPPLIER#,

O_SUPPLIER_NAME, O_ADDRESS. These properties could take any value in their domains of discourse. This mapping provides a unique facility, i.e., first to retrieve data from the database and then pass them onto ES as objects. ES triggers some rules based on the analysis of these objects and ultimately may cause an update in the database as a result of this triggering.

Table 2.1 : Translation table

DATABASE	KNOWLEDGEBASE
Entity type	Class
Entity	Object
Attribute	Property
Attribute value	Property value

Another aspect in comparing the two worlds for similarity, is the correspondence between the "relational join" in databases and "inheritance" in knowledge based systems or ES. The entities in an object-oriented knowledgebase are stored in a relationship called subtype-supertype hierarchy [Cohen 89]. For example, if object B is a subtype of supertype A, and A has some property P, then B inherits the property P. That is, if SUPPLIER has subtypes like WHOLESALE_SUPPLIER and RETAIL_SUPPLIER, and SUPPLIER has a property SUPPLIER#, then both the subtypes would inherit this property automatically. This "inheritance" finds its analogy to relational join operation in the database world. The join operation is performed to merge (not append) two relations (tables) on a common shared key (attribute column). Joining two relations SUPPLIER (SUPPLIER#, ADDRESS) and SUPPLY(SUPPLIER#, PART#, QUANTITY) on the common key SUPPLIER#, would yield the following single relation.

SUPPLIER_SUPPLY(SUPPLIER#, ADDRESS, PART#, QUANTITY)

The join relation is a binary operation and thus is capable of representing a subtype-supertype relationship. Comparing both the operations closely following observations can be made.

i) While inheritance is restricted to only subtype-supertype relationships, the relational join can capture any binary relationship.

ii) Inheritance is invoked automatically while join is invoked explicitly by giving the names of relations participating in the join operation.

iii) Inheritance is recursive. The subtypes may acquire properties from supertypes an arbitrary number of links above them in the hierarchy, but the number of relations in the join should be specified in advance.

Considering the facts about the similarities between database and knowledgebase systems, it is always better to integrate them into one (if there is a need) and have the power of both technologies.

2.3.2. The Integration

Examining both the worlds separately, one might be fascinated by trying to integrate both systems in one and have it named as either expert database system or logic database application. The inference engine and knowledgebase of the expert database system are responsible for deriving new data elements from primitive data using if-then rules and logical inference [Cohen 89]. The data items of database are divided into primitive and derived values. Primitive values are stored directly in the database and derived values are computed from primitive values. A typical architecture of an expert database system is shown in fig. 2.16. It should at least include the following components.

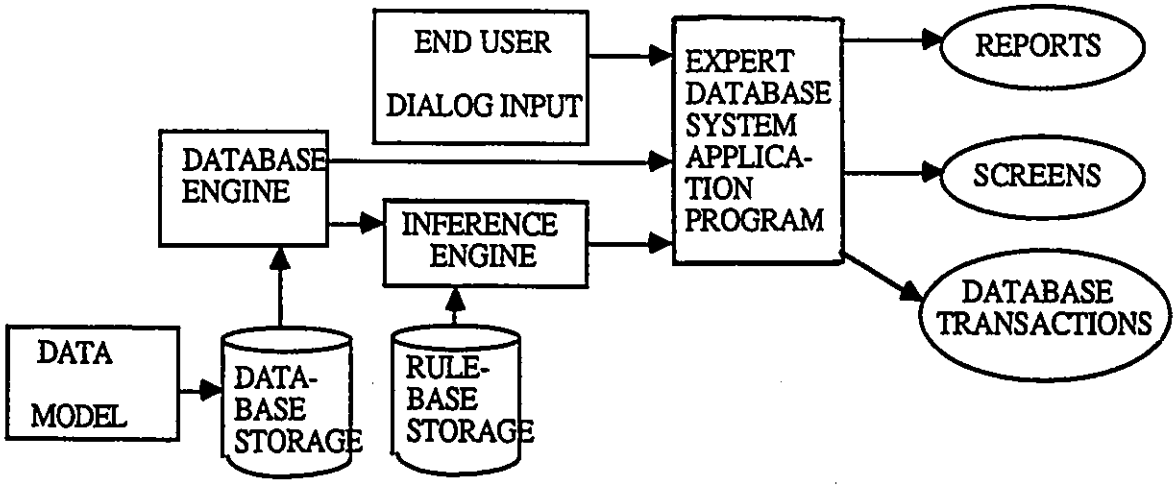


Fig. 2.16 Typical architecture of an expert database system

i) Expert database system application program - This module takes care of all the inputs and outputs and acts as an interface between the user and the system. It converts the user inputs into commands accepted by the system, as well as outputs formatted data which is the result of the user interaction. It makes explicit or implicit calls to the inference engine and/or database. A procedure call interface can be used to access the inference engine which gets converted into appropriate database access commands.

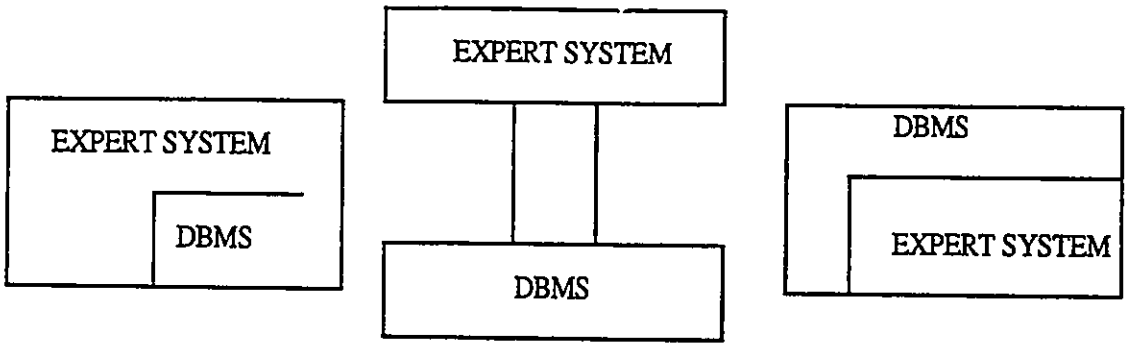
ii) Data model - it is stored in an active data dictionary.

iii) Database - it is stored and managed by a database engine.

iv) Rulebase storage and inference engine - they are part of the expert system which interacts with the database engine as well as the application program. The inference engine gets its input from the database and rulebase.

2.3.3 Technology of Integration

After examining the feasibility of integration of expert systems and databases, it is appropriate to discuss at this stage the spectrum of architectures for implementing it. The system organization may range from expert systems with integrated data management facilities, to the coupling of independent expert and database systems [Jarke 83]]. Fig. 2.17 shows three possible architectures for implementing an expert database system or logic database applications. Expert database system designers may opt for one configuration over another depending on data volume, multiplicity of data usage, data volatility, or data integrity and security requirements. The architecture shown in fig. 2.17 a) implements an expert system where data management functions are part of the ES. Using the architectures illustrated in fig. 2.17 b) and c) can effect database enhancements by the expert system. General strategies found in the literature are as follows.



a) Internal expert system database b) Coupling independent systems c) intelligent database system

Fig. 2.17 Basic architectures for ES - DBMS cooperation

2.3.3.1 Homogeneous System

The architectures of fig. 2.17 a) and c) pertain to homogeneous systems with integrated logic and DBMS. This approach integrates logic's inference mechanism and some DB functions into a single system. There can be two implementations of this approach [Leung 88, Sciore 88].

A. Pure logic system - A pure logic system consists of a logic programming language and few DB functions. The main purpose is to implement a first-order language. Prolog is a good candidate for such an implementation. Facts and rules are loaded into main memory prior to system execution. A query is answered by unification on the clauses stored in the memory. The overall system performance seems to be good but with the following drawbacks.

- Due to homogeneity of the storage structure, representing facts and rules in the same way makes it difficult to distinguish between them.
- The size of the database handled by the system is limited due to limitation in the main memory size.

This approach is good for applications which do not require large DB storage.

B. Enhanced logic system - These systems have DBMS functions built into a logic system or vice versa (fig. 2.17 a and c). The DB stored in secondary storage devices can be acquired any moment they are needed by the system. The data management functions are implemented in a

manner consistent with the logic system framework. The key issue here is to incorporate some of the powerful features of the DBMS functions into LS, for example, physical access techniques (such as indexing on first argument of the predicate), query optimization, and buffer management for effective DB system performance. Though these features are hard to incorporate in conventional logic programming languages, some of the languages like DEC20 Prolog do have indexing features, which decreases the search time.

But one might appreciate if the architecture is uniform and is written in one language, either an extended database programming language with deductive capabilities or an ES language with database/file management capabilities. Unfortunately, there are no languages available which could handle both ES operations and data management operations efficiently. Therefore, it is recommendable to use domain-independent expert systems which operates on an existing DBMS.

2.3.3.2 Heterogeneous System

Heterogeneous systems include loosely coupled independent DBMS and ES. Three candidate architectures can be distinguished depending on the main location of processing and type of interaction control.

A. Distributive processing and control. This strategy calls for a total distribution of processing and control. The two self-contained systems, i.e., ES and DBMS interact by exchanging messages. Whoever originates the communication is supposed to be the "master" and the receiver is the "slave". The advantage of this system is the transportability to other ES or DBMS. The bottleneck here is to decide how much each of the systems has to know in order to eliminate redundancy which causes serious problems of inconsistency and incompatibility.

B. Concentration of processing and control : The architectures based on concentration of processing and control have one of the subsystems assigned the more dominant role. It has one direction of flow of control and has a more variable distribution of labor compared to the first strategy. One of the major advantages is that the queries can be optimized but this is achieved at the expense of transportability. Another difficulty is that it integrates additional external subsystems.

C. Distributed processing with separate control : In this architecture, the processing is distributed but control is the responsibility of a separate subsystem, known as supervisor program. The supervisor program performs the operations for interfacing the ES with the DBMS, and manages the interaction between them. This architecture can be envisioned as a compromise architecture with the advantage of allowing for a smoother interaction with other subsystems. The difficulty is to build the supervisor that makes the full use of the capabilities of the constituent subsystems. All these approaches require an efficient mechanism for information passing to each other.

ES - DBMS communication is a vital organ of the aforementioned approaches. It has the responsibility of translating knowledge representations and transaction requests between subsystems. In a loosely coupled system, a communication channel allows data extraction from the existing database, and stores this "snapshot" as the expert system database. Data are extracted statically before the actual operation of the ES. The main disadvantage of this system is the non-applicability in cases where automated dynamic decisions, as to which database portion is required, are necessary.

Tight coupling supports dynamic decisions and keeps the communication channel open during expert system execution. But the tight coupling does need a high-level mechanism within the ES which raises questions about efficiency, since the element-by-element reasoning of the expert system must be connected with a set-oriented DBMS language.

All the communications require an optimization strategy for database references. This is done in the following steps.

i) A precompilation mechanism collects ES requests for data while simulating the ES deduction process.

ii) The collected database calls are optimized by simplifying and recognizing the common subexpressions.

iii) The optimized query is translated to the DBMS query language and executed by the DBMS. The database response is loaded as the internal expert system database. A garbage collection mechanism might be required to eliminate unnecessary data from internal expert system database time to time to yield higher performance.

2.3.4 Integration Tool

Once the technology for integration of ES and DB has been fixed, it remains to choose the particular implementation environment for both systems. Marriage of logic programming, which is a base for ES, and database technologies can provide a foundation for integrated, high-end reasoning systems. Each of the systems cannot provide efficient solution for the problems faced in each of them in isolation. The basic need for this marriage arises from the fact that most of the efficient inference engines (composite part of the ES) are poor in communicating and integrating themselves with existing software components. This added property of *connectivity* could give them an extra power and better source of knowledge, which is impossible to be encoded alone in the ES itself due to its large volume [Ketonen 89]. Data connectivity is a critical issue and in future one may assume transparent data access in much the same way as one gains transparent file access through network.

Two things should be kept in mind while integrating both systems. First, the language to build the ES and second, the language to access the existing data other than ES's own knowledgebase. As it is obvious that DBMS languages are good for data definition, retrieval, and updating, they cannot reason about the data. For that, it is required that the logical relationships between tuples and other entities are properly described. A DBMS embedded in any of the high level languages (suitable for logic programming) would perform the reasoning task very efficiently. The concern is not how to perform the computation, but what is to be computed. Logic programming provides a natural environment for reasoning about data. The major contribution of logic programming is its ability to define new and logical relationships between data by using simple statements called rules.

If relational database tables (relations) are compared for a similarity with the Prolog syntax, it is interesting to find out that both of them have more or less the same structure. For example, a relation contains several tuples (rows) and each tuple is divided into several attributes (columns). Similarly, in Prolog it can be represented as a number of clauses (equal to the number of tuples) with the same predicate name (same as relation name) and arity (number of arguments). Table 2.2 shows the comparison between a relational database and Prolog

[Rettig 87]. However, the basic difference between these two structures, one representing first order predicate calculus and the other the relational calculus, is the fact that the relational model does not put any restriction on the order of tuples in a relation but in Prolog a change in order in storing rules and facts may generate different results for the same query.

Aside from the differences in table 2.2, Prolog lacks properties for maintaining large databases as follows.

- Prolog seldom has an efficient way for accessing large DB.
- Knowledge (facts or predicates) cannot be shared among users though modules can be shared.
- Prolog cannot represent multiple views of relations or DB.

Table 2.2

RELATIONAL DATABASE	PROLOG
Base relation tuple	**Ground Clause (fact)**
- Constant number of fields	- Same predicate can appear with any number of arguments - Predicate can be defined with both facts and and Horn clauses
Attribute	**Predicate argument**
- Type restrictions	- No type restrictions
View definition	**Horn clause**
- View updated only when base relation can be updated	No distinctions between relations and views
Query	**Theorem**
- Queries evaluated over fixed interpretation	- Theorems evaluated according to proof theory making set oriented queries difficult.
Insert/delete	**Assert/retract**
- Semantics of ins/del operations on relations are part of the data model - Consistency is maintained across operations.	- Assert and retract apply equally to both facts and predicates. - No provision for maintaining consistency of facts with respect to existing predicates.

- Prolog cannot express database structures, i.e., hierarchy, network, or relationships.

But the following advantages of Prolog have something to offer to the DB world.

- The resolution theorem prover of Prolog is a powerful tool for a query.
- Prolog permits dynamic definition of DB entities and relationships.
- It allows recursively defined predicates for both data structure and queries.
- Some Prolog allow functions to appear as arguments contrary to DB where attributes are atomic.

2.3.4.1 The Mathematical Background for the Integration

Though the above discussions propose a Prolog - Relational DBMS integration, a theoretical background is required to see if this integration is feasible. The discussions here will be limited by showing some of the theoretical advancements made regarding this and details can be obtained in [Marque-Pucheau 83].

The answer to a query is basically the computation of the least model. A logic database is said to have a least model which is the intersection of all the models.

Definition - A *query* is a finite conjunction of units $B_1 \wedge ..... \wedge B_n$. An *answer* to a query is a ground substitution σ, such that, for each i ($1<=i<=n$) Bi σ belongs to the least model of the underlying logic database. Where a *ground instance of a Horn clause* H is a Horn clause obtained from H when all the variables have been substituted by constants.

Therefore, the computation of the least model gives the evaluation of the answer to a query. There are algorithms for defining the least model using equations the discussion of which is not the primary aim of this thesis. Basically the whole problem is to control the recursion and the optimization of the query. The algorithm should make sure that the recursion will end and the answer to the relational query will be obtained.

2.3.5 Implementations of Expert Database System

There have been several implementations of integrating ES with DBMS. It is not within the scope of this thesis to include them all. But some of them will be discussed here in order to appreciate the marriage of ES with DBMS and to know about the implementation issues for each of the designs.

2.3.5.1. SIENA : An ES - DBMS Interface for Network Management Control

There is a large demand for expert system and DBMS applications for modern telecommunication networks. The basic requirement is the capability of inferencing with a very large volume of factual information. SIENA (SQL Inference Engine for Network Applications) is being developed at the IBM Research Center. SIENA is a loosely-coupled approach of integrating an ES with existing DBMS [Whang 89].

Why networks need an expert system ?

A. Capability of inferencing with large factbase : In general, the data representing the status and operation of a large network are enormous (order of 106 facts). These data have to be stored in the secondary memory and the ES technology has to be extended for this kind of fact storage organization. Normally all the ES shells have main memory-resident knowledgebase. The large factbase can be managed by conventional DBMS. Following queries represent some of the examples representing the requirement of ES based network management.

a) To find a route between two nodes that support certain level of functions. For getting an answer for this query one has to know three things, i.e., information about physical connectivity, information about hardware and software that support the connectivity in each path, and capabilities of the hardware and software on each path. The execution of this type of query may require manipulation of thousands of facts.

b) The evaluation of the performance characteristics, like delay, throughput etc., also includes manipulation of huge amount of facts, gathered from different nodes and different customers on the network.

B. Capability of real-time response : An ES provides real-time services to adapt to dynamically changing network status.

C. Capability of sharing information with other network management tools : Since the networks are tending to have a central data repository, it is convenient and recommendable to let an ES manage this high level data repository. Other subsystems of the network flow data into the repository and the repository combines the similar data from different subsystems for inferencing on high level concepts.

Architecture and functions of SIENA

The overall architecture of SIENA is shown in fig. 2.18. This loosely coupled integration of ES and DBMS has a lot of advantages. The first is the time saved in designing a database by using an existing DBMS. The basic disadvantage of a loosely coupled system is the lack of a query optimization mechanism. Consider a rule in Prolog a(X,Y) <-- b(X,Z) & c(Z,Y) where there are thousand tuples satisfying the predicates b and c. 1001 DBMS calls will be required to evaluate a(X,Y). But if b and c are joined on the common variable Z then it will require only one database query. The problem of optimization in SIENA is solved by a smart loose coupling approach which performs optimization based on the following criteria.

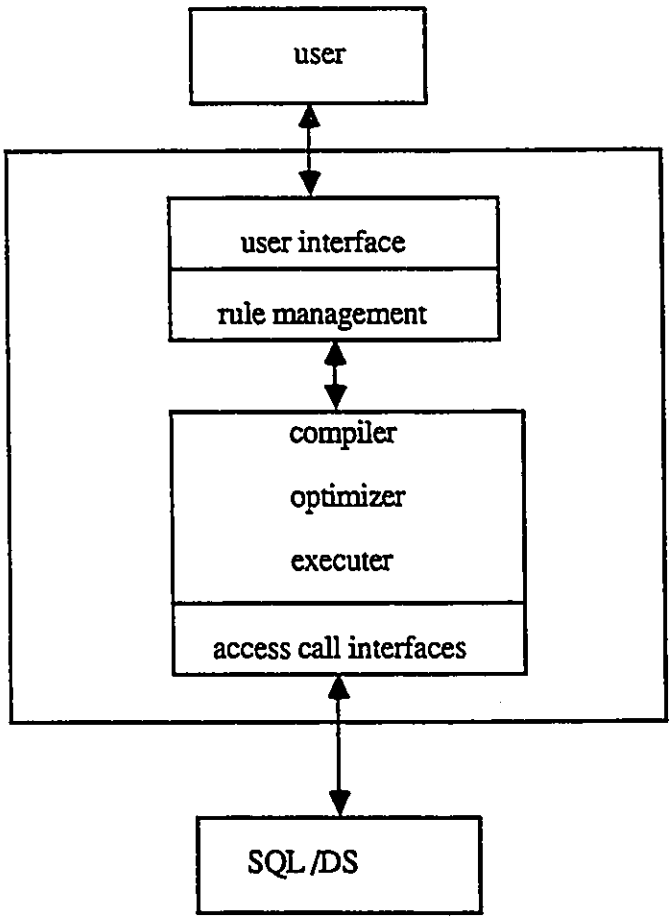


Fig. 2.18 The architecture of SIENA

i) reduce the database access.

ii) reduce the number of in-memory temporary relations.

iii) reduce the number of joins between in-memory and database relations.

SIENA provides a shell language based on function free Horn-clause logic that is extended for negation. The language supports recursion, negation, and disjunction. The system is connected to SQL/DS, a relational DBMS. The basic idea which is dealt in is the query optimization in a loosely coupled environment by reducing the number of DBMS calls [Whang 89] . The optimization states that if two or more predicates in a Horn-clause are having a common variable then first join those relations in the database and transfer the result to the remaining clauses or determine the output if no other clause is left. The optimization algorithm of SIENA also handles querying on recursive rules.

2.3.5.2 Efficient Database Access from Prolog

The design of interface between Prolog and relational database mentioned in [Ceri 89], has been proposed to enhance access performance to databases. The system uses memory-resident Prolog facts which are permanently stored in the secondary storage devices. The design proposes efficient accessing to database by never repeating the same query and by storing in main memory, in a compact and efficient way, information about the past interaction with the DBMS. This method enhances performance based on the past history of interaction rather than optimizing the query [Whang 89].

System Architecture

Most interfaces between Prolog and database use the paradigm of mapping each pattern matching operation between a Prolog predicate and the related facts to one query on the relational database [Whang 89]. But this system uses a new paradigm , in which information previously retrieved from the database is reused. The interface acts as a loading mechanism, retrieving information from the secondary storage and storing it into the main memory database. Then a Prolog interpreter can use the data stored in main memory in a conventional way. Fig. 2.19 represents the interaction between Prolog interpreter and the DBMS and the loading mechanism. The loading mechanism is a subsystem of the Prolog interpreter and is activated whenever a pattern

match is required for predicates whose facts are stored in secondary memory. The loading mechanism interacts first with the main memory to determine whether to access the secondary memory to bring the facts required. Once pattern matching is completed, the Prolog interpreter resumes execution. This interface can not only work with relational databases, but also with any file access system because the access to DB is at a lower level.

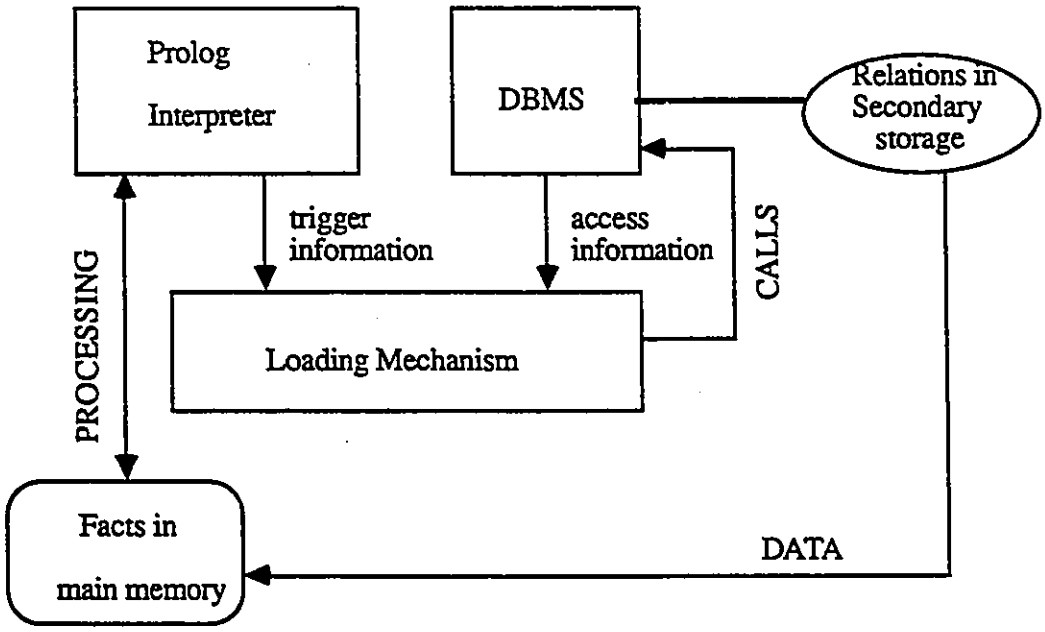


Fig. 2.19 Architecture of Prolog-DBMS interface

The major problem is to execute Prolog programs on the order of facts stored in DB. One of the research efforts has been made to design a language "Datalog", based on pure Horn clauses which also interfaces relational databases [Ullman 85]. That was an enhanced logic system design approach which is not viable commercially. The system proposed here proposes that the semantics of Prolog programs be retained and only the DB facts be insensitive to order. But once they are loaded into the main memory the order is maintained to ensure proper backtracking.

Operation

The following features have been considered in the design.

i) A relevant subset of the predicates of the Prolog programs have facts stored in secondary storage. They are recognized by the loading mechanism with the help of a dictionary which stores

the names of all database relations. The database predicates are not allowed to appear in the left-hand side of a rule.

ii) Order of facts in secondary storage is not relevant. This provides facility for any available access method. Order becomes relevant once the facts are stored in the main memory. Prolog also stores some facts and they precede the facts stored in the memory. This retains the conventional semantics of Prolog.

iii) Loading mechanism controls the page accesses and has information about cardinality of relations, size of attributes, number of values in a domain which are stored in the dictionary.

iv) The system only considers retrieval and not update.

```
p(X) :- rel(a,X,_,_,Y),t(Y).
q(X) :- rel(a,Z,X,X,T),j(X,Z),q(Z).
t(X) :- rel(b,X,Y,c,f).
```

REL				
A	B	C	D	E
a	b	c	b	c
a	a	c	b	d
c	a	c	b	d
c	a	d	f	d
a	c	c	b	c

Fig. 2.20 Example of interaction between prolog and a relational database

Fig. 2.20 shows a Prolog program and a database relation which corresponds to the database predicate (DBP) "rel" of the program. There are three occurrences of the DBP "rel" in the program and its first two occurrences in the rules p(X) and q(X) have the same variable bindings or in other words have the same static adornment. It means that in first two occurrences of "rel" the variable

bindings are as follows. (bound, free, free, free, free). Therefore, while executing the second occurrence of "rel", the loading mechanism does not have to load the data from the secondary memory because from the previous history (while executing the first occurrence of "rel") it knows that the data is already in the main memory and makes use of it. The approach discussed here is simple compared to that of discussed in [Whang 89], but saves unnecessary access to database which is not possible in the case of [Whang 89]. An algorithm has been proposed to perform this task. For more details of the algorithm one can refer to [Ceri 89, Ceri 87].

2.4 Natural Language Frontend to Query Systems

As it has been discussed in section 2.3 applications of expert database system require an user interface program for communicating with the user. The types of user may vary from highly skilled to common lay persons depending on the nature and the domain of the application. As it is obvious expert system programs are built to ease the work of an user who is not an expert of that field, i.e., AI software has all the expertise of a particular area built into it and is also meant to be used by a non-expert. But the bottleneck is the knowledge of the user about the structure of the data which can be recognized by the system. Mostly AI programs are developed using specific programming languages like Prolog, Lisp, Smalltalk etc. and the user is not aware of these language structures. Therefore, there should be some means for providing an interface between the user and the AI program. There are a lot of solutions available to this problem. The following sections contain a discussion regarding the strategies for building user-friendly interfaces and the emphasis will be on the use of natural language interfaces for such kind of applications.

The type of user interface an AI program needs is an adaptive one. The interface should be such that it could adapt itself to the current requirement of the user. One of the researchers warns, "unless we pay close attention to the user interface, users will become hopelessly lost and ineffective .. the complexity of the system should be transparent to the users" [Norcio 89]. The interface should be such that it would let the user and the interface feel that they are part of the system and are not external to the main system. The user models considered in the adaptive interface design should be based on the theories of cognition [Norcio 89,Kantorowitz 89]. The user interfaces can be divided into different categories based on their dialogue mode structures like

menu based systems, command line oriented systems, linguistic systems etc. The natural language interface (NLI) belonging to the category of linguistic systems, can be one of the candidates which can be implemented as a high level user interface to an expert database system. The following section takes a look into the justification of use of NLI in expert database systems.

Natural language interfaces (NLI) for solving problems, reasoning, communicating with databases etc. are another addition to the research and development achievements of engineers to pave an easy way in communicating with computers in natural language [Wallace 84]. This research and development trend on one hand helps to offload the burden from users learning different conversational languages and commands for different systems or applications, but on the other hand makes the design and implementation of the system achieve sometimes unparalleled complexity.

Though implementing a natural language interface seems to be a vast and cumbersome job, nevertheless researchers tend to build even more complex systems. Due to the vast domain of natural language understanding , the implementations are always constrained by a subset of it. That subset is basically the union of most commonly used structures of the language and application domain structures. The structures could be phrases and/or tokens (words). Natural language systems (NLS) belong to the category of adaptive user interfaces.

Adaptive user interfaces include a knowledgebase which represents the following [Norcio 89,Rissland 84] :

i) knowledge of the user

ii) knowledge of the interaction

iii) knowledge of the task domain

iv) knowledge of the system

i) Knowledge of the user - The knowledge of user is defined by modeling the user in conformity with the rules of cognitive psychology. There are two issues that arise in building the user model : determination of the information to be incorporated into the model and configuration of the model. Several strategies are used to construct and modify user models. The easiest technique for building user models is to classify users as novices and update their status to experts as they demonstrate more proficiency. One of the ways is provided by conceptual modeling.

Conceptual modeling : Conceptual modeling for linguistic interfaces is viewed as a subfield of database management and is based on semantic data models [Pilote 87]. The real world objects are related via relationships and there exist a bijective correspondence between objects in the model and real world entities. The conceptual modeling offers three abstraction mechanisms, namely

a) aggregation - grouping of components into a higher level concept.

b) representation - accounts for the relationships between objects and their instances.

c) generalization - provides a way to express constraints that define some concepts as refinements of other more general ones.

For example, the network shown in fig. 2.21 can provide an instant NLI for a new domain. Three types of semantic objects : entities (ENTITY_NAME), concepts (CONCEPT_NAME) and properties (PROPERTY_NAME) can be viewed respectively as specializations of the more general English lexical concepts PROPER_NOUN, CONCEPT_NOUN and RELATION_NOUN.

ii) Knowledge of the interaction - It is the domain of knowledge of interaction which controls the acceptability of natural language structures. NLIs possess many problems like it requires the user to adapt by restricting the legal inputs that are allowed. Following are the criteria for a user-friendly and usable natural language system for novices as well as experts.

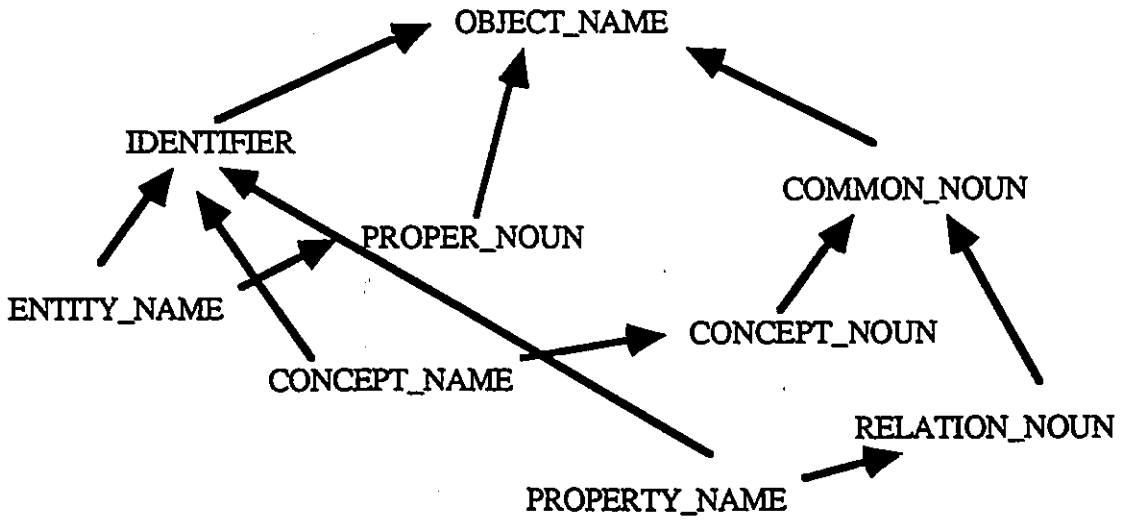


Fig. 2.21 Modeling of linguistic structure

A. It should be able to parse dialogue syntactically.

B. The interface should contain a rich semantic knowledge of the domain to resolve ambiguities..

C. It should be able to handle misspellings, missing punctuations etc.

D. The interface should provide maximal help to the novice user and be unobtrusive to the expert.

E. The interface should be partially transportable, if not fully to other domains.

iii) Knowledge of the task/domain - Knowledge of the task domain is needed by the interface to help it recognize the semantics in the dialogue structure. It consists of semantic parsing rules on the recognizable set of input phrases.

iv) Knowledge of the system - The knowledge of system keeps information about system limits and capabilities.

Our concern would be to engage ourselves only with the knowledge of interaction and task domain to design an expert database system based on natural language dialogue structures.

Though it has been always controversial and part of debate that NLS cannot satisfy the user requirements fully and the users get sometimes frustrated by the wild responses of the system [Rich 85], it has always been a challenge for the researchers to come up with better NLS. There have been many discussions on the viability of NLS and they have been compared with menu based and other restricted systems [Thompson 84]. The conclusion drawn is that for restricted applications dealing with a comparatively small domain of NL, it is useful to provide the human-computer-human communication in natural language format [Thompson 84,Sanford 88].

Therefore, the goal of providing an adaptive and intelligent user interface for our application, which is supposed to be using a smaller subset of the English language, is supported by the fact mentioned above. Other supporting criteria for justifying the choice of intelligent free format NLI for our application according to Rich [Rich 85] are as follows.

Cost : The cost factor in NLS is usually high but again it depends on the nature of application. The use of a comparatively smaller subset of English and the dialogue structures expected from the user, keeps the cost down and further the cost is maintained low by not including the database in

the intelligent interface module but rather by letting it to be managed by the database management system (DBMS) . This dynamicity will be discussed in detail in later sections.

Ease of learning : The choice of NLI is obviously suitable because of its ease of learning. The user does not have to learn specific commands and languages to communicate with the system. This criterion should be one of the prime considerations when the application is oriented towards non-technical users.

Precision : The precision of the system depends on the acceptability of the dialogue structures of the users. The results show that the precision of the system is low for a new user and keeps rising as the user gets acquainted with it more and more.

Conciseness : The user does not have to type big structures to communicate with the database. It would take longer time to form the same query in menu based NLS [Thompson 84].

Similarity between English and database query language : One of the major considerations for having a NLI was instigated by the similarity of English phrase structures and database query language commands. It was a better option to provide an intelligent NLI to the database rather than teaching the non-technical users about the commands and technicality of the system.

It is always better to incorporate an expert system for database applications to order to make the interface more user friendly and intelligent [Jarke 83]. Therefore, NLI is not interfaced directly to the DBMS, instead it communicates with an expert system which

i) generates queries for the DBMS (extracted from the NL request)

ii) guides the user to communicate appropriately by correcting errors and giving hints

iii) outputs comments based on the result of database access

iv) learns about that database

This avoids the unnecessary access to database in case of errors in the nature of query. Therefore, the design provides not only a NL dialogue, but also an adaptive system which keeps the frustration level of the user low by learning from users about the knowledge it does not have.

Chapter 3

Design and Implementation of a Natural Language Based Expert Database System

3.1 Introduction

The University of Ottawa Medical Communications Research Center has developed a multimedia database system for radiology communications [Vital 89]. The multimedia database maintains text documents, digital images as well as audio reports. This database is used to archive all the information available about the patient, that includes the patient's personal data together with his/her digitized X-ray images annotated with audio reports. Since the multimedia database design and manipulation require technical knowledge of the system, it is always desirable to have a front-end user friendly interface to the system in order for it to be accessible by a lay user. The nature and the user environment of this application encouraged us in providing an intelligent interface so that the manipulation of multimedia data does not require any technical knowledge from the user. There are several ways for building such interfaces. One of these ways is a natural language based intelligent interface (NLI) to the multimedia database.

It is always attractive to incorporate an expert system for database applications in order to make the interface more user friendly and intelligent [Jarke 83]. Therefore, the NLI is not interfaced directly to the DBMS, but instead it communicates with an expert system which

i) accepts user queries in natural language (English)

ii) guides the user to communicate appropriately by correcting errors and giving hints

iii) generates queries for the DBMS (extracted from the NL request)

iv) outputs comments based on the result of database access

v) learns about that database.

This avoids the unnecessary access to the database in case of errors in the nature of the query. Therefore, the design provides not only a NL dialogue, but also an adaptive system which keeps the frustration level of the user low by learning from users about the knowledge it does not have.

3.2 Overview of the System Components

The system organization shown in fig. 3.1 (a) -(b) is divided basically in four modules, i.e.,

i) the NLI,

ii) the query analyzer and generator,

iii) the result analyzer (or feedback to the system),

iv) and the DBMS incorporating the multimedia database.

The NLI is responsible for recognizing input queries in English and outputting the results produced by the co-operative efforts of expert system and DBMS [Shaw 90]. The expert system handles the reasoning, learning as well as generating queries for DBMS and answers the queries based on the database access results. We will discuss further the constituent components of each of the modules in detail.

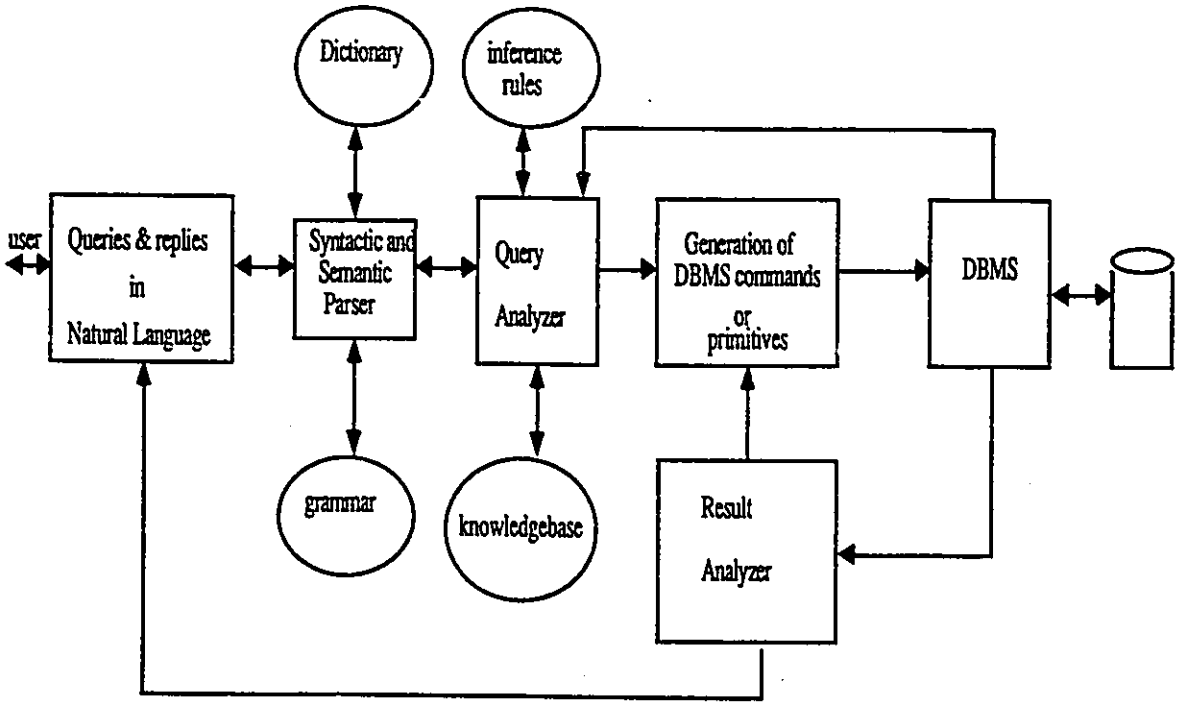


Fig. 3.1 (a) Block diagram of natural language based expert database system

3.2.1 Natural Language Interface

Many expert system modules and database querying systems developed in recent years use natural language structures as user input. Some of them are IRUS [Bates 84], IR-NLI [Brajnik86], TEAM [Grosz 83], EUFID, QPROC [Wallace 84] etc.

The natural language front-end to our expert database system supports a small subset of English structures and can be rated as sufficiently powerful for that domain of application [Appendix A]. It does not have the capability of differentiating among all the 22 request categories as described by Sanford & Roach [Sanford 88]. But it does recognize the structures belonging to the following categories :

i) Strongly demanding, e.g., "Give me the list of patients."

ii) Moderately demanding, e.g., "Who are the patients ? "

iii) Simply requesting, e.g., "Please, show me the list of patients."

iv) Abstract assertion, e.g., "list of patients" or "patients".

The interface consists of two modules, i.e., the syntactic analyzer and the semantic parser.

A. Syntactic analyzer It checks the input query for its validity from the syntax point of view. The module performs the following functions :

i) Forms the list of tokens (words) in the input query.

ii) Checks the tokens of the input query phrase. For that purpose it takes the help of a global dictionary which includes the common English tokens augmented with the application specific tokens. The detailed formation of the dictionary is illustrated in fig. 3.2.

iii) First the analyzer matches the token with the data available in the basic dictionary. If the token is not found there, it is passed on to the expert system which creates a dynamic token depending on the position and type of the token in the input phrase.

iv) If both the basic dictionary and the expert system are not able to recognize the token, the system prompts for an error at that position of the phrase and the user can either correct or invalidate the whole query. The system also produces help for the correct phrase structure.

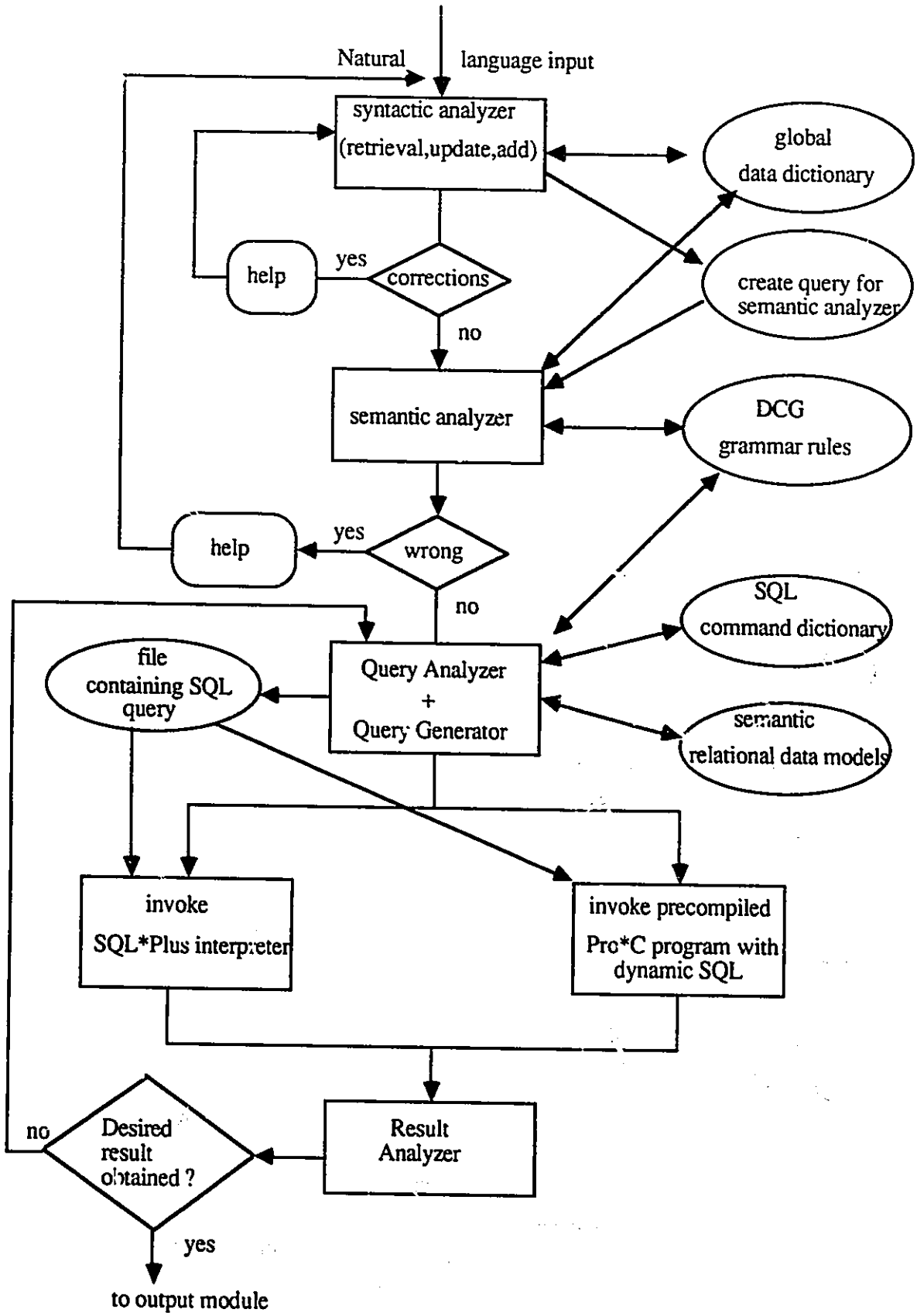


Fig. 3.1 (b) Interaction of program modules of the expert database system

v) The analyzer not only checks the tokens and matches them with the existing ones in the database, but it also uses the grammar rules to verify the structure and prompt for relevant information to the user.

Fig. 3.3 shows the syntax of the phrase structure defined by the grammar rules.

B. Semantic parser This module is responsible for extracting the information from the already validated by the syntactic analyzer query for the DBMS commands generating module of the expert system. The parser performs the following functions :

i) It uses the grammar rules to extract the required data from the input query.

ii) It calls the expert system to check the token which could not be matched with the data available in the global dictionary and had been created dynamically with a particular data type.

iii) It passes the semantic data collected from the input query on to the Query Analyzer and Generator (QAG) module .

iv) Along with the semantic data, it also passes the data type of the dynamic tokens to the QAG module. The data type of the dynamic token is determined by the parameters mentioned below.

a) its position in the phrase, and

b) its context.

The NLI design does not belong to the category of portable NLIs since the usage of portable NLIs has been abandoned due to restrictions imposed by the data modeling of specific applications. The design can be defined as partially portable, if at all it is required to use the interface for some other application, by adding additional information to the knowledgebase about the new system. To build a portable NLI is quite a hard task and is not worth designing since the NLIs are always application oriented. All the major NLI developments adopt this approach [Wilks75]. This NLI can be termed as topic dependent but structurally independent.

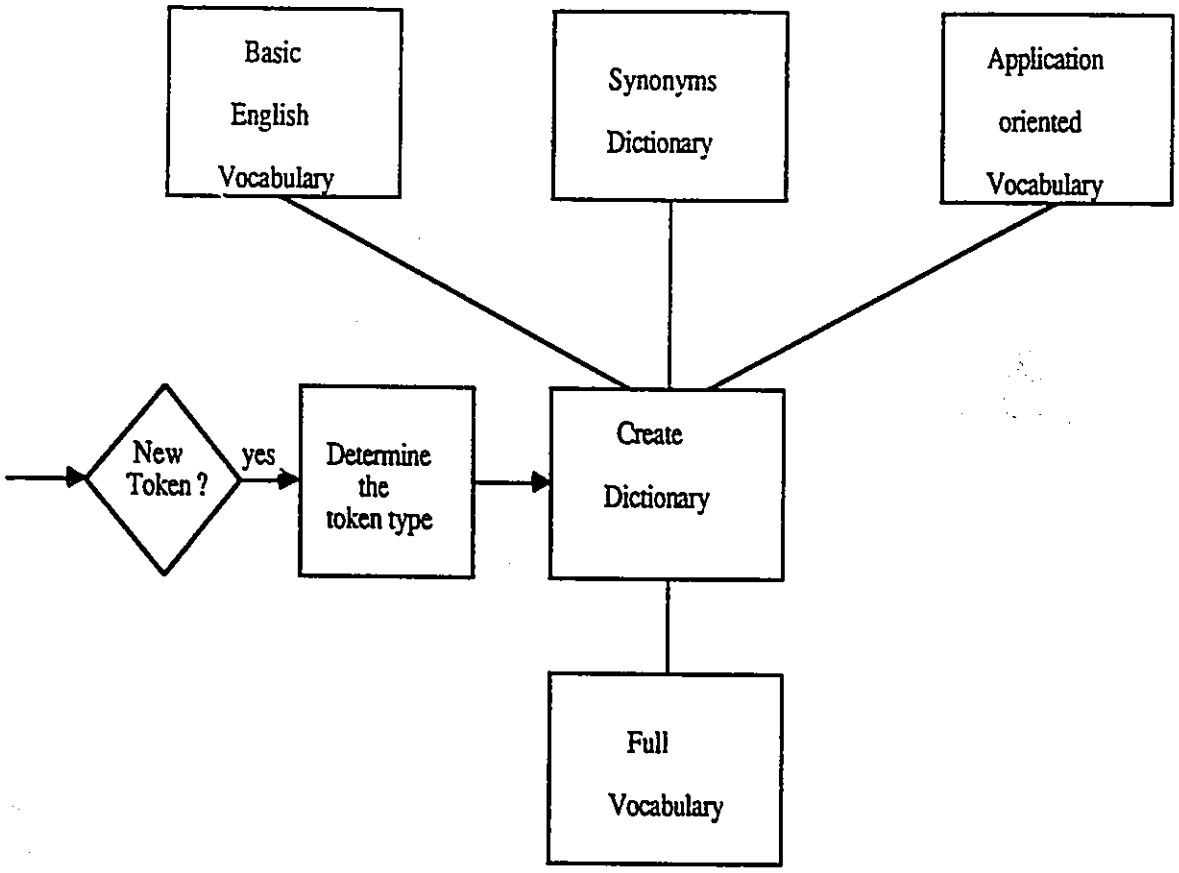


Fig. 3.2 Dictionary formation

Phrase :- <Subject > <Object> | <Object>

Subject :- <Article> <Noun_phrase> | <Noun_phrase>

Noun_phrase :- <noun> | <adjective> <noun> | <Interrogative>

Object :- <Verb_phrase> <Prepositional_phrase> | <Prepositional_phrase>

Prepositional_phrase :- <Noun_phrase> <preposition> <Noun_phrase> | <Verb_phrase> <Preposition> <Noun_phrase>

Prepositional_phrase :- <Prepositional_phrase> <connective> <Prepositional_phrase>

Verb_phrase :- <auxiliary_verb> <Verb> | <Verb>

Preposition :- of | in | on | with

Connective :- and | or

Interrogative :- what | when | where | who

Fig. 3.3 Syntax definition of the input phrase structure

3.2.2 Query Analyzer & Generator (QAG) Module

The QAG module is the heart of the system. On one hand it generates commands for the DBMS while on the other hand produces formatted results of the query, prompts and help for the user engaged in an interactive session, in case there are some descrepancies in the input query [Appendix A]. The module is designed as a rule based system. It includes the following.

i) A knowledgebase containing facts about the data stored in the multimedia database. It actually does not contain the data themself as opposed to some database query systems like QPROC [Wallace 84].

ii) An inference engine which governs the interaction with the user by firing rules, based on certain situations. Result of these firings is the interaction with the user who in turn has the power to change the situation dynamically.

Since the actual data is managed by the DBMS, the knowledgebase does not include them. But the inference engine does include the rules and the knowledgebase which help to generate the DBMS commands in SQL. It interacts with the semantic parser for getting the information to generate the SQL commands. The module performs the following functions :

a) Retrieves the semantic data from the semantic parser.

b) Analyzes the semantic data in context with the semantic relational data model, available in its knowledgebase.

c) Builds a corresponding SQL command structure which is the most qualified for the data available from the input query.

d) Writes the SQL command structure in a shared among ES and DBMS area.

e) Invokes DBMS for execution of the command written in the common area.

f) Gets the results of the command execution by the DBMS and further analyzes them on the following criteria :

i) If the DBMS responds to the query, then the resulted data is passed onto the result analyzer module.

ii) In case of any error in the query it tries other alternative SQL commands which could also correspond to the input query. It also tells about the modification it made for providing the result.

iii) Step (ii) continues exhaustively until there is no alternative command or DBMS comes out with the correct result.

The module has the capability of handling the dynamic tokens of the input query and mapping them to suitable entity structures for the DBMS depending on the data type of the token. It creates a dynamic token with a particular data type determined by the above facts. This is needed by the semantic parser to generate data for the DBMS commands generating module. Components which the module interacts with are shown in fig. 1 (b) .

3.2.3 Result Analyzer

This comparatively small module is responsible for analyzing the result of the execution of the SQL query by DBMS [Appendix B]. If the result is not compatible with the desired output then the module passes the relevant parameters like error codes and system messages to the query analyzer which again depending on the acquired information either tries to reformulate the query, if possible, or report adequate messages to the user.

3.2.4 Database Management System

The database management system (DBMS) is an independent module which stores and manages the multimedia database [Appendix B]. The DBMS provides two distinct services : Radiology Information Database (RID) and Multimedia Storage System (MSS). RID handles the overall patients' medical information whereas MSS stores and manages unformatted data such as images, voice and text. The contents of the MSS are linked logically with their corresponding contents of RID via logical descriptors. The Data Definition Language (DDL) defines the structure of the relations, attributes and dictionary entries and the Data Manipulation Language (DML) is used to store, retrieve, and manipulate the data stored in the database.We have chosen to use a widely accepted relational DBMS, namely ORACLE, for our application. RID and logical descriptors are managed by ORACLE DBMS and MSS is managed by the Unix File System. ORACLE is based on a query language SQL which includes both data definition and data manipulation languages. Fig. 3.4 shows the Multimedia Database Server architecture. The Multimedia Server can be accessed by several remote workstations through the local area network (LAN). Each workstation runs the user application software. The expert database system module is part of this software providing the user a better and intelligent access to the DBMS. The communication between the workstations and the server is handled by TCP/IP. The queries can be sent to the server to get executed and the results of this execution can be obtained from it.

Fig. 3.5 contains the Entity - relationship (E-R) model of the radiological database used for the application. It contains the entities, e.g., DOCTORS, PATIENT_FOLDER, ROOMS, MEDICAL HISTORY, REPORTS, and ITEMS. The entity ITEMS is more general in the sense that it consists of subsets such as IMAGES, VOICE, TEXT, and DATA segments. The E-R model does not contain the corresponding attributes for the entities in order not to complicate the model. Depending on the above E-R model the database schema has been defined and the corresponding relations and their attributes of RID are shown in fig. 3.6.

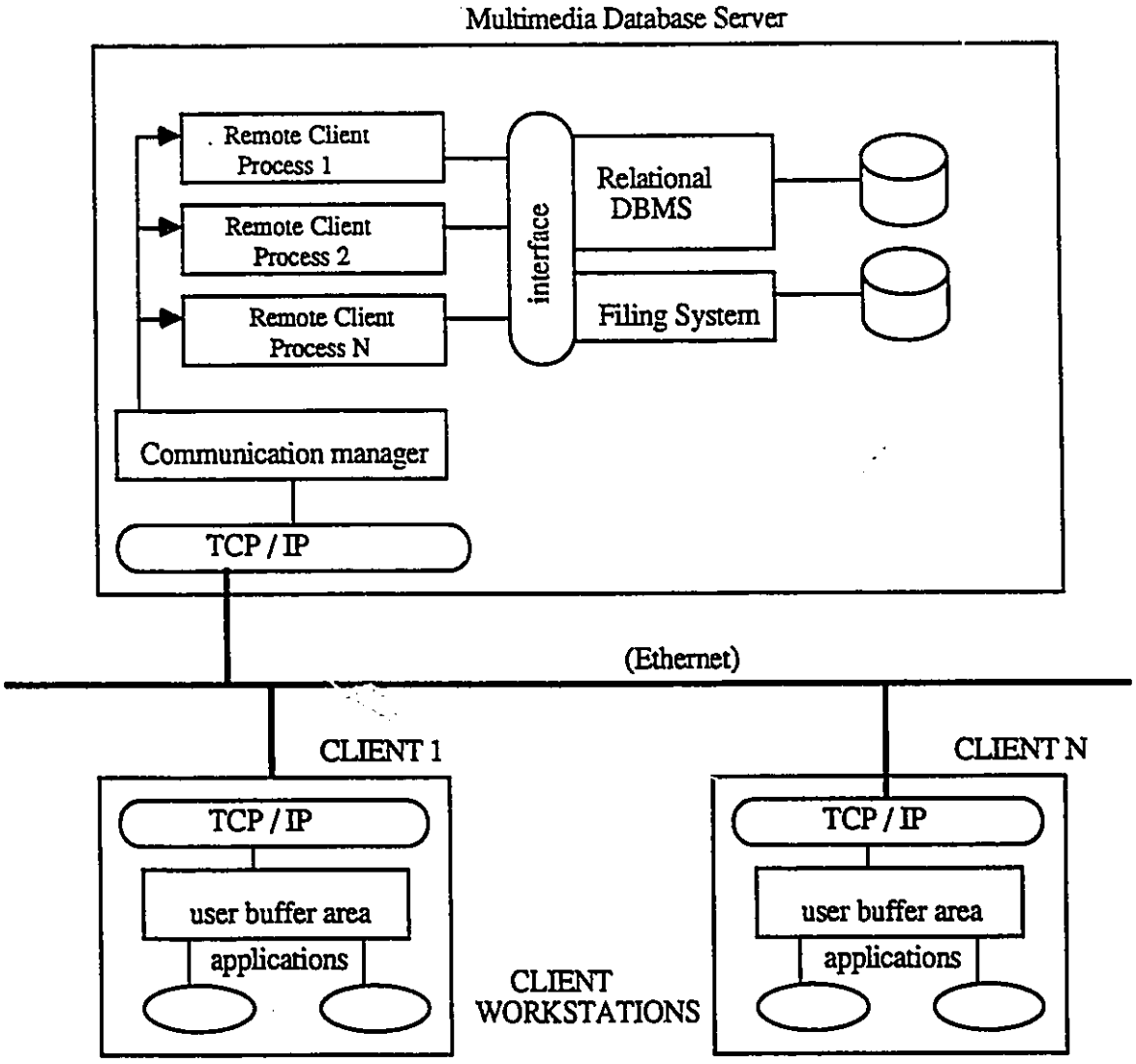


Fig. 3.4 Multimedia database server architecture

The DBMS accepts the commands in SQL from the command generating module. It executes the SQL query and returns the control to the expert system which further processes the available data either to output to the user or to invoke some other interface (for acquiring image and/or voice annotations). Other than returning the values of the data queried, it also passes values of the system parameters to help ES to interpret them into a suitable answer back to the user. The database is protected from the integrity and consistency point of view.

3.3 Data Modeling

Data modeling is required to help map the semantics of the phrase presented in natural language into corresponding meaningful database query commands which when executed by DBMS would fetch the results expected by the NL query. Queries are not limited to data retrieval , in fact they could be handling data manipulation as well as storage.

Conceptual data modeling allows to extract semantics hidden in the natural language phrase [Stallord 84, Pilote 87]. Conceptual modeling always deals with semantic models which represent, in the simplest of their form, two real world "objects" related by some means. Semantic data modeling is application oriented and varies with the domain of application. Data modeling for the NLI depends on the phrase structures and rules of English grammar which helps the expert system to generate corresponding DBMS queries. Before defining the data model for the application one has to determine the power of the grammar implemented, i.e., the type of sentences which can be recognized by the grammar. All NLI applications use a limited set of grammar rules. In our case we have defined the types of sentences recognized by the grammar in a previous section.

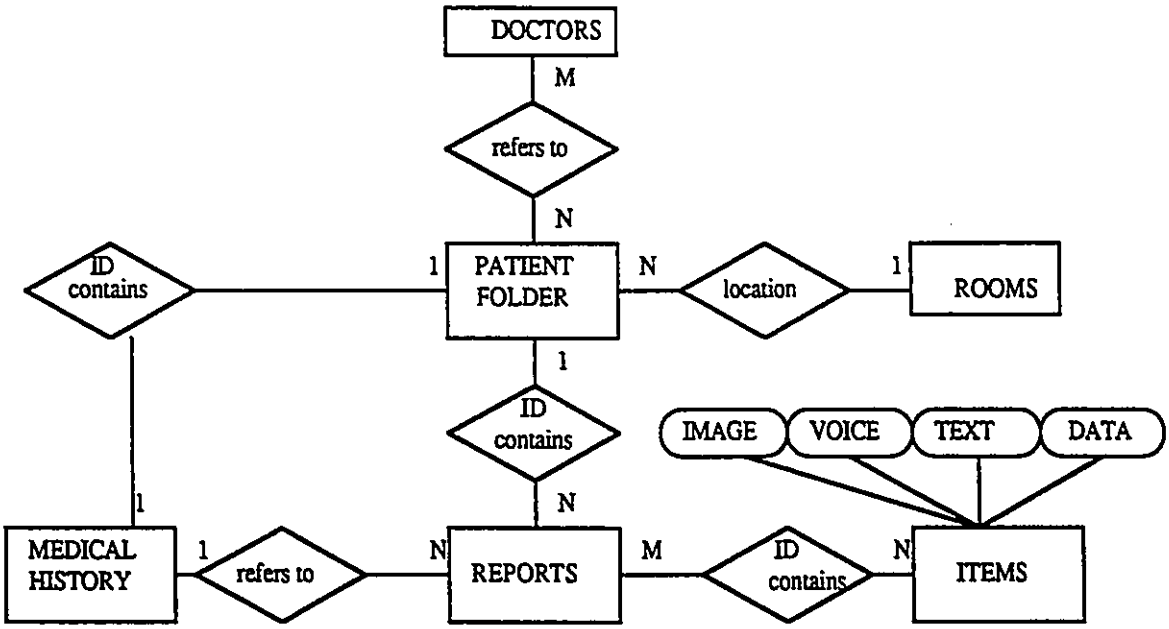


Fig. 3.5 The Entity - Relationship Model of the radiological database

(courtesy [Vital 89])

PATIENT_FOLDER (Folder_Unique_No, Patient_Name, Patient's_Social_Insurance_No, Physician, ...)

ARCHIVED_REPORTS (Report_Reference_No, Folder_Unique_No, Examination_No, Diagnosis, ...)

IMAGES (Image_Unique_No, Report_Reference_No, Image_Date, Adress, Width, Height, Area, ...)

Fig. 3.6 Database relational schema

3.3.1 Grammar

Fig. 3.7 shows the structure of the semantic parser. Actually the basic idea of having a NLI front end to the DBMS is based on the fact that the structure of English sentences representing NL queries has a close similarity to the structure of corresponding DBMS commands required to represent the NL query. For example, a query in English may look like as follows.

List the patients of Dr. Hansen

Whereas the SQL command to represent the same query would look like as :

```
SELECT   patient_name
FROM     patient
WHERE    doctor_name = 'Hansen'
```

The information required for the SELECT and WHERE clauses is mentioned in the NL query itself and can be recognized by the grammar rules. But it has to be augmented with the information for "FROM clause" which requires a semantic data modeling.

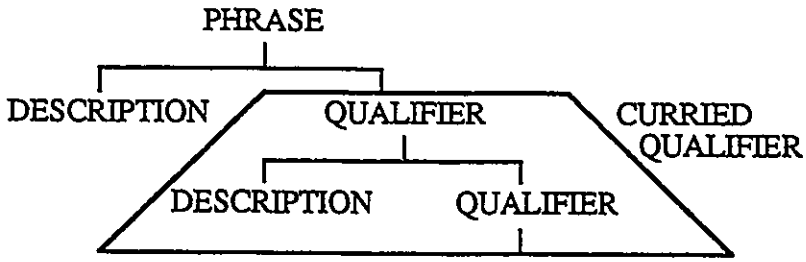


Fig. 3.7 Structure of the semantic parser

3.3.2 Modeling

Semantics can be drawn from the NL query with the help of noun phrases, verbs and prepositional phrases. For example, consider the following queries which expect the same outcome.

1: What is the location of Mr. Smith ?

2: Show the room no. of Mr. Smith.

3: Where is Mr. Smith located ?

4: Room no. of Mr. Smith.

These are four different ways of asking the same question. In the first sentence the semantics is drawn from the tokens "what", "location" and "Mr. Smith". The second sentence has its meaning hidden in the tokens "show", "room no." and "Mr. Smith". Whereas sentence no. 3 contains meaningful tokens like "where", "Mr. Smith" and "located". The fourth sentence is an abstract assertion which hides its meaning in "room no." and "Mr. Smith".

Now the grammar is used by the expert system to extract the meaningful tokens from the input phrase. Note that all four input phrase should generate the same SQL query. The syntax of the structure provides the following.

a) In assertive sentences the first noun phrase gives the information required by the SELECT clause and the prepositional phrase supplies data for the WHERE clause. Then the expert system refers to the data models as shown in fig. 3.8 to determine the corresponding relations and attribute names for generating the FROM clause as well as supplying the additional information to two other clauses.

b) In interrogative phrases the interrogative token (word) tells about the entity being looked for. This is achieved from the semantic networks as shown in fig. 3.9. Then either the noun phrase or the verb decides precisely the target.

c) In abstract assertive sentences it is easier to derive the semantics from noun and prepositional phrases.

The points discussed above make it obvious that two approaches, i.e., semantic relational models and semantic networks have been implemented together to draw the semantics from the input query. Referring to fig. 3.9 for example, "where" and "what" lead to the object class "location" and the noun "room no." and verb "located", which are instantiations of the same object class, confirm the entity being referred to. Instantiations like location, located, room no. etc, refer to the same entity which stores its different properties co-relating them with each other.

Since the knowledgebase of the expert system does not contain any data regarding the actual occurrences of the tuples or the attributes of the relations, a question arises as to how these instantiations (basically proper nouns and numbers) can be incorporated into data models. Actually the expert system, depending on the structure of the phrase, the grammar rules applied to it and its context, determines the data type of the proper noun which maps itself into corresponding relational attributes. For example, Mr. Smith will be recognized from its context as a patient name and not a doctor's name.

3.4 Implementation

The implementation of the natural language based expert database system conforms to the technique discussed in [Whang 89]. This specifically defines the interface of expert systems to relational databases with the help of Prolog [Ceri 89]. The expert system is required for interfacing with a large amount of data stored in secondary storage and managed by a relational DBMS namely ORACLE. This kind of interface is known as loosely coupled since the data is managed by existing DBMS.Since Prolog structures have correspondence to the expressions of relational algebra, it has been proved to be a good tool for such interfaces and applications [Marque-Pucheau 83, Ceri 89].

The syntactic analyzer and semantic parser have been written in MPROLOG on a SUN workstation. The expert system which handles the dynamicity of the queries, has a knowledgebase of semantic data models to generate DBMS commands and also has sufficient knowledge to answer back to the user the results of the query or any error in the presentation. Its inference engine contains the rules for defining the structure of the DBMS query. The modules of ES are written in MPROLOG. On one side the module has been interfaced to the semantic parser module while on the other it communicates with DBMS. Fig. 3.10 illustrates the examples of query types which can be recognized by the system. The queries are categorized into simple and composite queries. At present the expert database system answers all the simple queries and part of composite queries. The queries like "Is John married ?" or "How many patients suffer from heart disease ?" cannot be answered by the design implemented. The answer to the first query should either be yes or no which depends on the data retrieved from DBMS. For answering complex queries the expert system has to be augmented with the corresponding intelligence.

Relation	attributes	domain	Rel.Attribute
patient_of	name	DOC_NAME	PATIENT.DOC_NAME
doctor_of	patient	person	Person
in_location	location	LOC_NAME	PATIENT.LOC
location_of	patient	Person	Person
admitted_on	patient	Person	Person
admission_date	admin_date	DATE	PATIENT.CDATE
with_status	patient	Person	Person
status_of	status	Status	ARCHIVED.PSTAT
exam_date	patient	Person	Person
next_exam_date	exam_date	DATE	ARCHIVED.NDATE
diagnosis_of	patient	Person	Person
	diagnosis	DIAGNOSIS	ARCHIVED.DIAG
	exam_no	EXAM	ARCHIVED.EXAM
Person	Lastname	NAME	PATIENT.LNAME
	Firstname	NAME	PATIENT.FNAME
	id	ID	PATIENT.ID
image_of	patient	Person	Person
	image	image	IMAGES.
	exam_no	EXAM	IMAGES.IUN, IMAGES.RUN

Fig. 3.8 Semantic data models

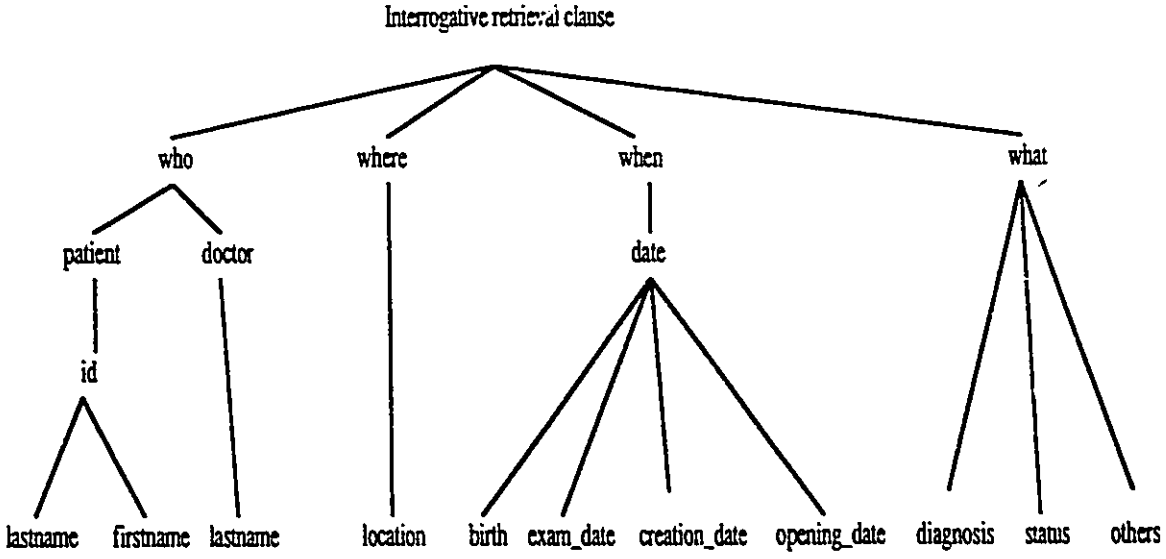


Fig. 3.9 Semantic network for interrogative clauses

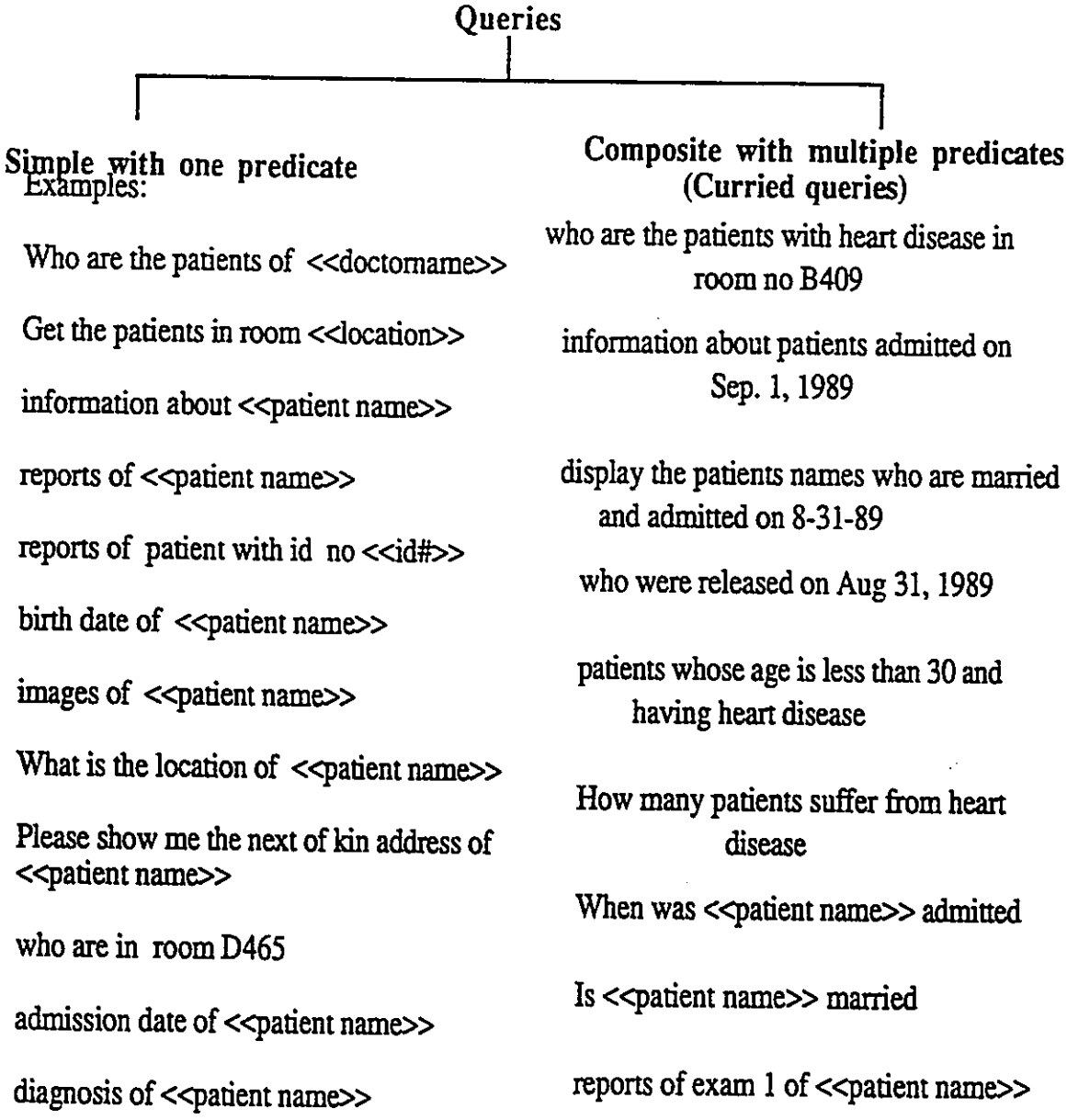


Fig. 3.10 Query examples

Chapter 4

Enhancements of the Implementation

The basic structure of the implementation of a natural language based expert database system has been discussed in the previous chapter. As the expert system program interacts with the database software it is recommendable to provide a smoother interface between them which should be transparent to the user. For the user, each of the modules of the expert database system should work as an integrated part of the whole system. The interfacing of the expert system written in MPROLOG to the ORACLE can be done in two ways, as follows.

4.1 Execution of Queries with the Help of Interpreter

The expert system may call the SQL interpreter and provide it with the SQL query to get it executed immediately. This approach has the advantage of executing the SQL query immediately but does not provide any information about the system parameters corresponding to various situations, e.g., mode of execution, error codes etc. Following contains some of the drawbacks.

i) The interpreter takes longer time in compiling and executing the query and the time might be significant if more alternative SQL queries for the same original request is generated in case of intermediate failures.

ii) The interpreter does not have the capability of formatting the output data.

iii) It does not return the error codes. To obtain them one has to explicitly call the corresponding OCI (ORACLE Call Interface) module.

The first version of the interface was based on the the SQL interpreter and immediate preparation and execution of the SQL queries. But the above constraints and desire for building a truly feedback oriented expert database system prompted us to exploit the power of expression provided by dynamic SQL.

4.2 Using Dynamic SQL

The dynamic SQL interface of ORACLE provides facility for the SQL statements to be embedded in a C program as well as take care of dynamically changing queries. The latter is more important to our requirements, where the number of the set of queries generated by the expert system can be very significant due to the vast nature of the radiology application and its users. It also provides some system parameters which can be accessed by the programmer and has a well defined interface to the error handling routines. The following illustrate the advantages of using dynamic SQL for the radiology application.

i) It provides an effective though complicated way of accessing the system parameters of the ORACLE routines which handle the higher level parameters of the SQL queries.

ii) It facilitates the interface with the SQL error handling routines which are useful for the feedback purposes for the application.

iii) By virtue of getting access to the system parameters which define the queries, the programmer need not use other formatting utilities like SQL*Form for formatted data outputting. This is a major advantage over the PREPARE and EXECUTE model.

iv) The queries need not consist of host variables for containing the actual result of the query execution.

v) And lastly, it is worth mentioning that, since the SQL commands for this interface are embedded in a C program and are precompiled at the time of being invoked by the expert system, it takes lesser amount of time to execute the same query as compared to the interpretive model. The execution time is a major consideration in data retrieval systems since it increases with the volume of data stored.

The following section discusses in brief the interface which handles the dynamically defined SQL statements.

4.2.1 Structure of the Dynamic SQL Interface

The dynamic SQL is used when the structure of the database query to be executed is not known until the program is executed. For the radiology application the expert system generates a wide variety of database queries just to satisfy one single request from the user. Therefore, the nature of SELECT and WHERE clauses cannot be predetermined and taken care of in the program. The dynamic SQL interface provides some cursors which can be associated with the SQL queries and determines a descriptor and communication areas SQLDA and SQLCA for the communication and allocation purposes respectively. Fig. 4.1 illustrates the SQLDA structure which is useful in accessing the system parameters associated with the SQL queries and fig. 4.2 depicts the parameters of the error detection and recovery module of SQL communication.

Attributes of SQL descriptor :

N - Descriptor size in number of entries (for memory allocation purposes).

V - Contains the address of each of the main variables of the query.

L - Holds the length of buffer for each column to be projected.

T - Contains the data type of the buffer (character string / number / date etc.).

I - Contains addresses of indicator variables.

F - Holds number of columns to be projected.

S - Points to the variable names used in the query.

M - Points to the maximum lengths of each of the variable names.

C - Contains current lengths of each of the var. names.

X - Points to indicator var. name pointers.

Y - Points to max. lengths of indicator variables of the query.

Z - Points to the current lengths of indicator variables of the query.

Fig. 4.1 The structure of SQL descriptor

The dynamic SQL descriptor together with the error detection and recovery module provide a greater flexibility and sophistication to the expert database system on the whole. The results of the query execution and the data provided by these interfacing modules are used by the expert system for backtracking and helping it to perform as a truly feedback expert database system.

Parameters of SQLCA :

sqlcode - holds zero when the execution is successful.

is positive when no tuple is found.

becomes negative if there is an error or a system failure.

sqlerrmc - contains text corresponding to the error number.

sqlerrml - holds the length of the text in sqlerrmc.

sqlwarn - is an eight-byte array and each of the bytes indicates warnings for different conditions.

First byte is zero when no warnings are set.

Fig. 4.2 Parameters of SQL communication an error detection interface

Chapter 5

Sample Runs and Results

The following shows some examples of sample runs and the results produced. After the user provides his/her name and password the expert database system is invoked. The corresponding database queries shown in square braces are not seen by the user. It has been illustrated for clarity.

>> Please enter your query !

>> Show me the folder no. of Smith.

[SELECT FUN
FROM PATIENT
WHERE PATIENT.LNAME = 'Smith']

FOLDER NO. - 12345

>> When is the birthdate of Mr. Wilson.

[SELECT BDATE
FROM PATIENT
WHERE PATIENT.LNAME = 'Wilson']

BIRTHDATE - 31-7-1945

>> Who is the doctor of Williams.

[SELECT DNAME
FROM PATIENT
WHERE PATIENT.LNAME = 'Williams']

DOCTOR NAME - Kragen

>> What is the diagnosis of Smith.

[SELECT DIAG
FROM PATIENT REPORT
WHERE PATIENT.LNAME = 'Smith' AND PATIENT.FUN = REPORT.FUN]

DIAGNOSIS - Heart disease

>> When is the next exam date of Robinson.

[SELECT NDATE
FROM PATIENT REPORT
WHERE PATIENT.LNAME = 'Robinson' AND PATIENT.FUN = REPORT.FUN]

NEXT EXAM DATE - 15-5-1990

>> When was Johnson admitted.

[SELECT ODATE
FROM PATIENT
WHERE PATIENT.LNAME = 'Johnson'

OPENING DATE - 8-12-1989

>> What is the social insurance number of patient with id 23456.

[SELECT SIN
FROM PATIENT
WHERE PATIENT.PID = '23456']

SOCIAL INSURANCE NO. - 907465654

>> Please tell me the address of patient in room no. 405.

[SELECT LNAME PADD
FROM PATIENT
WHERE PATIENT.PLOC = '405']

NAME - Johnson
ADDRESS - 67 Henry st., Ottawa

>> Can you tell me the condition of Mrs. Carson.

[SELECT PSTAT
FROM PATIENT REPORT
WHERE PATIENT.LNAME = 'Carson' AND PATIENT.FUN = REPORT.FUN]

STATUS - Critical

>> How many images of report 1. of Smith.

[SELECT COUNT(IUN)
FROM PATIENT REPORT IMAGES
WHERE PATIENT.LNAME = 'Smith' AND
PATIENT.FUN = REPORT.FUN AND REPORT.REPREF = IMAGES.REPREF]

IMAGES - 3

>> Display the image 1 from report 3 of Johnson.

[SELECT IADD
FROM PATIENT REPORT IMAGES
WHERE PATIENT.LNAME = 'Johnson' AND REPORT.REPREF = '3' AND
IMAGE.IUN = '1' AND
PATIENT.FUN = REPORT.FUN AND REPORT.REPREF = IMAGES.REPREF]
{/* Then the image acquisition moule can be invoked by providing parameter values like IADD obtained from IMAGES */}

>> Show me the patients of Dr. Kragen in room 305.

[SELECT LNAME FNAME
FROM PATIENT
WHERE PATIENT.DOCNAME = 'Kragen' AND PATIENT.PLOC = '305']

NAME - Smith Andrew

>> show me the adress of Mrs. Carson E.

* Please check the syntax of the word adress.

* Do you want me to correct it ? (y/n)

* It could be address. Now accessing the database

[SELECT PADD

FROM PATIENT

WHERE PATIENT.LNAME = 'Carson' AND PATIENT.FNAME = E*]

ADDRESS - 67 Henry St., Ottawa

>> information on Mr. Williams

[SELECT *

FROM PATIENT

WHERE PATIENT.LNAME = 'Williams']

{ After accessing the data the program finds that there is no data on Williams then it prompts for it and displays the corresponding data on lastnames starting with 'W' }

There is no data on Williams. I will show you other relevant data stored.

LNAME - Wordsworth

.

.

MARITAL STATUS - Married

The interaction shown in the above examples gives a clear description of the expert system - database integration. The time required for execution and producing the results varies in a wide range. it actually depends on the following.

i) The volume of data stored in the database.

ii) The no. of "joins" considered for the equivalent query.

iii) The correctness of the intermediate results, i.e., the number of times alternative queries are generated and executed for the same original request.

Chapter 6

Conclusion and Future Work

In this thesis an effort has been made to design and implement a natural language based expert multimedia database system for providing the user with an intelligent user-interface. Various data modeling techniques and inferencing mechanism were used for implementing the design. Results obtained from the implementation show that the design is powerful regarding the NL queries which closely resemble the SELECT - FROM - WHERE structure of the DBMS query. So is the case with most of the existing proposals. Effort has been made to enhance the system so that it would recognize the composite queries discussed in chapter 3.

Since the domain of application is quite specific, a user gets along with the system very rapidly. We now consider enhancing the NL based expert database system for retrieval of image and voice data to make it a truly expert multimedia database system. As per now it can be tailored to interface itself to the image acquisition module to retrieve radiological images from the secondary storage databases.

6.1 Achievements

i) A natural language (English) based expert database query system has been designed and implemented.

ii) The expert system software is written in MPROLOG.

iii) The knowledge in ES has been represented by a semantic relational model, semantic networks and production rules.

iv) The ES converts English queries into correspondingSQL queries, checks the validity of input queries, corrects the errors in the input query, analyzes the result and outputs relevant data to the user.

v) A program has been developed in C with embedded SQL commands to interface with the ES module. This C program has its own independent capability of controlling the program execution.

vi) A feedback mechanism connected through the results of the SQL query execution has been implemented for analyzing the query execution and making certain decisions on the query preparation or backtracking and finding other alternatives in case of ambiguities or incorrect formulation of the query.

vii) The expert database module can be interfaced to the image acquisition module in order to retrieve images requested by the user.

6.2 Future Work

Some of the points discussed below can be considered in the future to enhance the system capability.

i) The system can be made more powerful by incorporating more composite queries other than those discussed in chapter 3.

ii) With a significant effort the system could be made to respond to the queries which expect answers in "yes" or "no", e.g., "Has Johnson left the hospital ?"

iii) The semantic parser module can be enhanced to prompt for the syntax errors and correct them on the basis of the context of the incorrect tokens.

iv) The module should be extendable to voice annotations and it would be desirable for the expert database system to be interfaced with another expert image retrieval module, which is seen by some experts as a future perspective for multimedia radiology communications.

Once again it is worth mentioning that the design gives a free hand to the lay user and is very useful for this kind of applications.

References

[Ambron 88] S. Ambron, " Introduction to Multimedia ", *Interactive Multimedia, ed. by S. Ambron and K.Hooper*, Microsoft Press, Washington, 1988, pp 1-11.

[Andriole 85] S.J. Andriole, "*Applications in AI* ".Petrocelli Books Inc. USA, 1985.

[Barr 81] A.Barr and E.A.Feigenbaum, " *The Handbook of Artificial Intelligence* ", William Kaufmann Inc., 1981.

[Bates 84] M. Bates, " Accessing a Database with Transportable Natural Language Interface ", *Proc. of First Conf. on AI Applications, Denver, Dec. 1984*, pp 9 - 12.

[Bertino 88] E. Bertino et al., " Document Query Processing Strategies : Cost Evaluation and Heuristics ", *Proc. of Conf. on Office Information System, Palo Alto, March 1988*, pp 169 - 181.

[Bing 88] J. Bing, " Conceptual Text Retrieval for Legal Information Retrieval Systems ", *Proc. of RIAO 88, Cambridge, MA, March 1988*, pp574-585.

[Blank 89] G.D.Blank, " A Finite and Real-time Processor for Natural Language ", *Communications of the ACM*, v. 32, no. 10, Oct.1989, pp 1174 - 1189.

[Blumberg 89] R.E.Blumberg and D.H. Walters, " A PC Based Multimedia Document Manager ", *IEEE Journal on Selected Areas in Communications*, v.7, no.2, Feb. 1989, pp 283 - 289.

[Boguraev 83] B.K.Boguraev and S.Jones, " A NAtural Language Front-end to Databases with Evaluative Feedback ", *Proc. of ICOD-2 Workshop on New Applications of Databases, Churchill College, England, Sept. 1983*, pp 159 - 182.

[Borkin 80] S.A. Borkin, " *Data Models : A Semantic Approach for Database System* ", MIT press, 1980.

[Brajnik 87] G.Brajnik et al., " Design and Implementation of IR-NLI : An Intelligent user Interface to Bibliographical databases ", *Expert database Systems, L.Kerschberg ed.*, The Benjamin Cummings Publishing Company Inc., 1987, pp 151 - 162.

[Ceri 87] S.Ceri, et al. " Interfacing Relational Databases and Prolog Efficiently ", *Expert Database Systems, L.Kerschberg ed.*, The Benjamin Cummings Publishing Company Inc., 1987.

[Ceri 89] S. Ceri et al., " Efficient Database Access from Prolog ", *IEEE trans. on Software Eng.*, v.15, no.2, Feb. 1989, pp 153 - 164.

[Chen 76] P.P.S. Chen, " The Entity-Relationship Model - Toward a Unified View of Data ", *ACM Transactions on Database Systems*, vol. 1, no. 1, 1976, pp 9-36.

[Cohen 89] B. Cohen, " Merging Expert Systems and Databases ", *AI Expert*, Feb. 1989, pp 22 - 31.

[Croft 88] W.B.Croft and R.Krovetz, " Interactive Retrieval of Office Documents ", *Proc. of Conf. on Office Information System, Palo Alto, March 1988*, pp 228 - 235.

[Date 86] C.J.Date, " *An Introduction to database Systems* ", Addison Wesley Pub. Company, 1986.

[Du 89] D.H-C. Du et al., " An Efficient File Structure for Document Retrieval in the Automated Office Environment ", *IEEE trans. on Knowledge and Data Eng.*, v.1, no.2, June 1989, pp 258 - 273.

[Eirund 88] H.Eirund and K.Kreplin, " Knowledge Based Document Classification Supporting Integrated Document Handling ", *Proc. of Conf. on Office Information System, Palo Alto, March 1988*, pp 189 - 196.

[Fox 88] E.A. Fox et al., " Implementing a Distributed Expert-Based Information Retrieval System ", *Proc. of RIAO 88, Cambridge, MA, March 1988*, pp 708 - 726.

[Gauch 88] S.Gauch and J.B.Smith, " Intelligent Search of Full-Text Database ", *Proc. of RIAO 88, Cambridge, MA, March 1988*, pp 162 - 171.

[Goldberg 89] M. Goldberg et al., " A Multimedia Medical Communication Link Between a Radiology Department and an Emergency Department ", *SPIE Medical Imaging III : PACS System Design and Evaluation, vol. 1093, Jan/Feb 1989*, pp 307 - 317.

[Gross 88] D. Gross, " Applications of Semantic Networks in Database Search and Retrieval ", *Expert Systems*, v. 5, no.2, May 1988, pp 152 - 154.

[Grosz 83] B.Grosz, " TEAM : A Transportable Natural Language Interface System", *Proceedings of Conference on Applied Natural Language Processing, Santa Monica, 1983*, pp 39-45.

[Grudin 89] J. Grudin, " A Case against User Interface Consistency ", *Communications of the ACM*, v. 32, no. 10, Oct. 1989, pp 1164 - 1173.

[Hahn 88] U.Hahn and U.Reimer, " Automatic Generation of Hypertext Knowledgebases ", *Proc. of Conf. on Office Information System, Palo Alto, March 1988*, pp 182 - 188.

[Harris 85] M.D. Harris, " *Introduction to Natural Language Processing* ", Reston Publishing Company, 1985.

[Idrissi 88] J.Idrissi, " Self-Structured Data Banks Semantic Integrity and Query Assistance ", *Proc. of RIAO 88, Cambridge, MA, March 1988*, pp 439-449.

[Jarke 83] M.Jarke and Vassiliou, " Databases and Expert Systems : Opportunities and Architectures for Integration ", *Proc. of ICOD-2 Workshop on New Applications of Databases, Churchill College, England, Sept. 1983*, pp 185 - 201.

[Kantorowitz 89] E. Kantorowitz and O. Sudarsky, " The Adaptable User Interface ", *Communications of the ACM*, v. 32, no. 10, Oct. 1989, pp 1352 - 1358.

[Karmouch 89] A.Karmouch and N.D.Georganas, " Multimedia Document Architecture for Medical Applications ", *Proc. of SPIE Medical Imaging III : PACS System Design and Evaluation, 1989*, pp 113 - 121.

[Karmouch 90] A.Karmouch, L.Orozco-Barbosa, N.D. Georganas and M.Goldberg, " A Multimedia Medical Communication System ", *IEEE J. on Selected Areas of Communication*, April 1990.

[Ketonen 89] J.A. Ketonen, " Toward Reasoning ", *AI Expert*, Feb. 1989, pp 44 -49.

[Kluzniak 85] F.Kluzniak and S. Szpakowicz, " *Prolog for Programmers* ", Academic Press, 1985.

[Leung 88] Y.Y.Leung and D.L.Lee, " Logic Approaches for Deductive Databases ", *IEEE Expert*, Winter 1988, pp 64 - 75.

[Li 84] D.Li, " *A Prolog Database System* ", Research Studies Press, 1984.

[Marcus 86] C. Marcus, " *Prolog Programming : Applications for Database Systems, Expert Systems and Natural Language Systems* ", Addison Wesley, 1986.

[Marque-Pucheau83] G. Marque-Pucheau et al., " Interfacing Prolog and Relational Database Management System ", *Proc. of ICOD-2 Workshop on New Applications of Databases, Churchill College, England, Sept. 1983*, pp 225 - 244.

[Minker 88] J. Minker, " *Foundations of Deductive Databases and Logic Programming* ",Morgan Kaufmann Publishers, 1988.

[Minsky 68] M. Minsky, " *Semantic Information Processing* ", MIT Press, 1968.

[Napier 89] H.A. Napier et al., " Impact of a Restricted Natural Language Interface on Ease of Learning and Productivity ", *Communications of the ACM*, v.32, no.10, Oct. 1989, pp 1190 - 1198.

[Newell 72] A. Newell and H.A.Simon, " *Human Problem Solving* ", Prentice Hall, Englewood Cliffs, N.J., 1972.

[NewquistIII 88] H.P.Newquist, " Intelligent Data : Getting to First Base, *AI Expert*. Jan. 1988, pp 21 - 23.

[Norcio 89] A.F.Norcio and J.Stanley, " Adaptive Human - Computer Interfaces : A Literature Survey and Perspective ", *IEEE trans. on Systems, Man, and Cybernetics*, v. 19, no.2, March/April 1989, pp 399 - 408.

[Pernici 88] B.Pernici, " Supporting OIS Design through Semantic Queries ", *Proc. of Conf. on Office Information System, Palo Alto, March 1988*, pp276-283.

[Pilote 87] M. Pilote, " Modeling Linguistic User Interfaces ", *Expert database Systems, L.Kerschberg ed.*, The Benjamin Cummings Publishing Company Inc., 1987, pp 399 - 412.

[Pitrat 88] J. Pitrat, " *An AI Approach to Understanding Natural Language* ", North Oxford Academic Publishing, 1988.

[Ramsey 89] C.L.Ramsey and V.R.Basili, " An Evaluation of Expert Systems for Software Engineering Management ", *IEEE trans. on Software Eng.*, v.15, no.6, June 1989, pp 747 - 759.

[Rettig 87] M.Rettig, " Marrying Logic Programming and Databases ", *AI Expert*, June 1987, pp 15 -19.

[Rich 85] E.Rich, " Natural Language Interfaces ", *Readings in Human-Computer Interaction : A Multidisciplinary Approach, ed. by R.M. Baecker and W.A.S. Buxton*, 1985, pp 442 - 450.

[Risch 88] T. Risch et al., " A Functional Approach to Integrating Database and Expert Systems, *Communications of the ACM*, v. 31, no.12, Dec. 1988, pp 1424 - 1437.

[Robinson 90] R. Robinson, " The Four Multimedia Gospels ", *Byte*, vol. 15, no. 2, Feb. 1990, pp 203 - 212.

[Sanford 88] D.L.Sanford and J.W.Roach, " A Theory of Dialogue Structures to Help Manage Human - Computer Interaction ", *IEEE trans. on Systems, Man, and Cybernetics*, vol. 18, no.4, July/August 1988, pp 567 - 574.

[Schnupp 87] P.Schnupp and L.W.Bernhard, " *Productive Prolog Programming* ", Prentice Hall, 1987.

[Schur 88] S.Schur, " The Intelligent Database ", *AI Expert*, Jan. 1988, pp 26-34.

[Schwartz 87] S.P. Schwartz, " *Applied Natural Language Processing* ", Petrocelli Books Inc. USA, 1987.

[Sciore 88] E.Sciore and D.S.Warren, " Integrating Databases and Prolog ", *AI Expert*, Jan. 1988, pp 38 - 44.

[Shaw 90] G.K.Shaw, A.Karmouch and N.D.Georganas, " Design and Implementation of a Natural Language Based Expert System for a Radiology Multimedia Database ", *Canadian Conference on Electrical and Computer Eng., Sept. 1990.*

[Shelter 90] T. Shelter, " Birth of the BLOB ", *Byte*, vol. 15, no. 2, Feb. 1990, pp 221 - 226.

[Simmons 84] R.F.Simmons, " From Menus to Intentions in Man-Machine Dialogue ", *Proc. of First Conf. on AI Applications, Denver, Dec. 1984*, pp 2 - 8.

[Stallard 84] D. Stallard, " Data Modeling for Natural Language Access ", *Proc. of First Conf. on AI Applications, Denver, Dec. 1984*, pp 19 - 24.

[Stonebraker 89] M.Stonebraker, " Future Trends in Databse Systems ", *IEEE trans. on Knowledge and Data Eng.*, v.1, no.1, March 1989, pp 33 - 44.

[Tanimoto 86] S.L.Tanimoto, " *The Elements of Artificial Intelligence : An Introduction Using LISP* ", Computer Science Press, 1986.

[Thompson 84] C.W. Thompson, " Recognizing Values in Queries or Commands in a Natural Language Interface to Database ", *Proc. of First Conf. on AI Applications, Denver, Dec. 1984*, pp 25 - 30.

[Ullman 85] J.D.Ullman, " Implementation of Logical Query Languages for Databases", *ACM trans. on Database Systems*, vol. 10, no. 3, Sept. 1985, pp 289-321.

[Ullman 88] J.D.Ullman, " *Database and Knowledgebase Systems* ", vol. 1. Computer Science Press, 1988.

[Vital 89] D. Vital et al., " Multimedia Data Management in Radiology Communications ", *Proc. of Canadian Conference on Electrical and Computer Eng., Montréal, Sept. 1989*, pp 590 - 593.

[Wallace 84] M. Wallace, " *Communicating with Database in Natural Language* ". John Wiley and Sons, 1984.

[Whang 89] K.-Y. Whang and S. Brady, " High Performance Expert System - DBMS Interface for Network Management and Control ", *IEEE Journal on Selected Areas in Communications*, v.7, no.3, April 1989, pp 408 - 417.

[Yang 86] C-C. Yang, " *Relational Databases* ", Prentice Hall, Englewood Cliffs, N.J., 1986.

Appendix A

```
/* This module parses the input queries syntactically as well as semantically and generates DBMS
   queries. It also analyzes the result. */

module input.

all_global.

import   (article / 1,auxiliary / 1,noun / 1,preposition / 1,
        sentence / 1,verb_copulative / 1,verb_transitive / 1).

/*$eject*/
body.

go :-
    repeat, print("<<Ready>>"), nl, read_token(C), cut_input,
     (((C==bye;C==expression end)), print("Have a good day !");
      start(C), fail) .

start(C) :-
    set_state(symbolic_var_handling,on), process(C), ! .
start(C) :-
    print("Question not understood, please try again !"), nl .

process(C) :-
     read_line(C,LINE,C2,_),
      sentence(LINE,RES,ATT,REL1,REL2,REL_ATT,VAL,SUC), SUC=y,
      print(RES), nl, set_channel(outfile(1),name="query.sql"),
      set_output(outfile(1)),
      sql_query(ATT,REL1,REL2,REL_ATT,VAL),
      close_output(outfile(1)), host('dyna') .
```

```
sql_query([ATT1],REL1,[],REL_ATT,VAL) :-
    print("SELECT "), print(ATT1), nl, print(" FROM "),
     print(REL1), nl, print(" WHERE "), print(REL_ATT),
     print(" = "), print(VAL), nl .
sql_query([ATT1,ATT2],REL1,[],REL_ATT,VAL) :-
     .t("SELECT "), print(ATT1), print(","), print(ATT2), nl,
     print(" FROM "), print(REL1), nl, print(" WHERE "),
     print(REL_ATT), print(" = "), print(VAL), nl .
sql_query([ATT1],REL1,REL2,REL_ATT,VAL) :-
    print("SELECT "), print(ATT1), nl, print(" FROM "),
     print(REL1), print(","), print(REL2), nl, print(" WHERE "),
     print(REL_ATT), print(" = "), print(VAL), print(" AND"),
     nl, print("PATIENT.FUN=DIAGNOSIS.FUN"), nl .
sql_query([ATT1,ATT2],REL1,REL2,REL_ATT,VAL) :-
    print("SELECT "), print(ATT1), print(","), print(ATT2), nl,
     print(" FROM "), print(REL1), print(","), print(REL2), nl,
     print(" WHERE "), print(REL_ATT), print(" = "), print(VAL),
     print(" AND"), nl, print("PATIENT.FUN=DIAGNOSIS.FUN"),
     nl .

read_line(C,LINE,C2,_) :-
    not word(C), !, read_token(C1), cut_input,
     rest_of_line(C1,STRING,C2), same(LINE,[C|STRING]) .

rest_of_line(C,[C|STRING],C2) :-
    not word(C), !, read_token(C1), cut_input,
     rest_of_line(C1,STRING,C2) .
rest_of_line(C,[],C) .

same(LINE,STRING) :-
    LINE=STRING .

word(expression end) .

:- go .

endmod /* input */ .
```

```
module parser.

all_global.

export   (article / 1,auxiliary / 1,noun / 1,preposition / 1,
       sentence / 1,verb_copulative / 1,verb_transitive / 1).

/*$eject*/
body.

dynamic(name/1).

sentence(LINE,RES,ATT,REL1,REL2,REL_ATT,VAL,y) :-
   syntax(LINE,RES), assertive(RES,ATT,REL1,REL2,REL_ATT,VAL) .
sentence(_,_,_,_,_,_,_,n) :- !.

syntax([H|T],[HR|RES]) :-
    not word1(H), substring(H,2,H1), make_char_list(H,[HE|TA]),
     convert_char(HE,CO),
     ((CO<58,add_statement(name(H)),HR=H;CO>64,CO<91,convert(H1
        ,lower_case,H2),concatenate(HE,H2,H3),look_again(H3,HR
        ))), syntax(T,RES) .
syntax([H|T],[HR|RES]) :-
    not word1(H), print("Please check the correctness of $'"),
     print(H), print("$' and replace it with a correct one !"),
     nl, read_token(H1), cut_input,
      ((word1(H1),HR=H1;substring(H1,2,H2),make_char_list(H1,[HE
        |TA]),convert_char(HE,CO),CO>64,CO<91,convert(H2,
         lower_case,H3),concatenate(HE,H3,H4),print(H4),nl,HR=
        H4,add_statement(name(HR)))), syntax(T,RES) .
syntax([H|T],[H|RES]) :-
    word1(H), syntax(T,RES) .
syntax([],[]) .

look_again(H3,HR) :-
    word1(H), HR=H3;
    HR=H3, add_statement(name(HR)) .
```

```
word1(W) :-
    article(W);
    noun(W);
    adjective(W);
    verb(W);
    auxiliary(W);
    preposition(W);
    pronoun(W);
    toobj(W) .

assertive(LINE,ATT,REL1,REL2,REL_ATT,VAL) :-
    adverbial_phrase(LINE,ATT,REL1,REL2,REL_ATT,VAL) .
assertive(VP,ATT,REL1,REL2,REL_ATT,VAL) :-
    verb_phrase(VP,ATT,REL1,REL2,REL_ATT,VAL) .
assertive([N|VP],ATT,REL1,REL2,REL_ATT,VAL) :-
    pronoun(N), verb_phrase(VP,ATT,REL1,REL2,REL_ATT,VAL) .

noun_phrase(ART,NOUN) :-
    article(ART), noun(NOUN) .

verb_phrase([V|ADP],ATT,REL1,REL2,REL_ATT,VAL) :-
    verb(V), adverbial_phrase(ADP,ATT,REL1,REL2,REL_ATT,VAL) .
verb_phrase([AUX,V|ADP],ATT,REL1,REL2,REL_ATT,VAL) :-
    auxiliary(AUX), verb(V),
     adverbial_phrase(ADP,ATT,REL1,REL2,REL_ATT,VAL) .

verb(V) :-
    verb_transitive(V);
    verb_copulative(V) .

adverbial_phrase([TOOBJ|ADP],ATT,REL1,REL2,REL_ATT,VAL) :-
    toobj(TOOBJ),
     adverbial_phrase(ADP,ATT,REL1,REL2,REL_ATT,VAL) .
adverbial_phrase([ART|ADV],ATT,REL1,REL2,REL_ATT,VAL) :-
    article(ART),
     adverbial_phrase(ADV,ATT,REL1,REL2,REL_ATT,VAL) .
adverbial_phrase([OBJ1|PREP],ATT,REL1,REL2,REL_ATT,VAL) :-
    noun(OBJ1), prep_phrase([OBJ1|PREP],REL_ATT,VAL),
     comp1(OBJ1,ATT,REL1,REL2) .
```

```
adverbial_phrase([ADJ,N|PREP],ATT,REL1,REL2,REL_ATT,VAL) :-
   adjective(ADJ), noun(N), prep_phrase([N|PREP],REL_ATT,VAL),
    comp2(ADJ,N,ATT,REL1,REL2) .
adverbial_phrase([ADJ,N1,N2|PREP],ATT,REL1,REL2,REL_ATT,VAL) :-
   adjective(ADJ), noun(N1), noun(N2),
    prep_phrase([N1|PREP],REL_ATT,VAL),
    comp3(ADJ,N1,N2,ATT,REL1,REL2) .
adverbial_phrase([N1,N2|PREP],ATT,REL1,REL2,REL_ATT,VAL) :-
   noun(N1), noun(N2), prep_phrase([N1|PREP],REL_ATT,VAL),
    comp2(N1,N2,ATT,REL1,REL2) .
adverbial_phrase([ADJ,PR,N1,N2|PREP],ATT,REL1,REL2,REL_ATT,VAL) :-
   adjective(ADJ), preposition(PR), noun(N1), noun(N2),
    prep_phrase([N1|PREP],REL_ATT,VAL),
    comp3(ADJ,N1,N2,ATT,REL1,REL2) .
adverbial_phrase([N1,of,N2|PREP],ATT,REL1,REL2,REL_ATT,VAL) :-
   noun(N1), noun(N2), prep_phrase([N2|PREP],REL_ATT,VAL),
    comp2(N1,N2,ATT,REL1,REL2) .

prep_phrase([physician,P|PREP],REL_ATT,VAL) :-
   preposition(P), val_phrase(physician,PREP,REL_ATT,VAL) .
prep_phrase([physicians,P|PREP],REL_ATT,VAL) :-
   preposition(P), val_phrase(physician,PREP,REL_ATT,VAL) .
prep_phrase([doctor,P|PREP],REL_ATT,VAL) :-
   preposition(P), val_phrase(physician,PREP,REL_ATT,VAL) .
prep_phrase([doctors,P|PREP],REL_ATT,VAL) :-
   preposition(P), val_phrase(physician,PREP,REL_ATT,VAL) .
prep_phrase([OBJ,P|PREP],REL_ATT,VAL) :-
   preposition(P), val_phrase(OBJ,PREP,REL_ATT,VAL) .

val_phrase(physician,[N|T],REL_ATT,VAL) :-
   name(N), REL_ATT="PATIENT.LNAME", concat_quote(N,VAL) .
val_phrase(patients,["Dr",N|T],REL_ATT,VAL) :-
   name(N), REL_ATT="PATIENT.DOCTNAME", concat_quote(N,VAL) .
val_phrase(patient,["Dr",N|T],REL_ATT,VAL) :-
   name(N), REL_ATT="PATIENT.DOCTNAME", concat_quote(N,VAL) .
val_phrase(patients,[N|T],REL_ATT,VAL) :-
   name(N), REL_ATT="PATIENT.DOCTNAME", concat_quote(N,VAL) .
val_phrase(patient,[N|T],REL_ATT,VAL) :-
   name(N), REL_ATT="PATIENT.DOCTNAME", concat_quote(N,VAL) .
```

```
val_phrase(OBJ,[N|T],REL_ATT,VAL) :-
    name(N), REL_ATT="PATIENT.LNAME", concat_quote(N,VAL) .
val_phrase(OBJ,[room,no,NUM|T],REL_ATT,VAL) :-
    name(NUM), REL_ATT="PATIENT.PLOC", concat_quote(NUM,VAL) .
val_phrase(OBJ,[room,NUM|T],REL_ATT,VAL) :-
    name(NUM), REL_ATT="PATIENT.PLOC", concat_quote(NUM,VAL) .
val_phrase(_,[id,no,NUM|T],REL_ATT,VAL) :-
    name(NUM), REL_ATT="PATIENT.ID", VAL=NUM .
val_phrase(_,[identification,no,NUM|T],RELATT,VAL) :-
    name(NUM), REL_ATT="PATIENT.ID", VAL=NUM .
val_phrase(_,[id,NUM|T],REL_ATT,VAL) :-
    name(NUM), REL_ATT="PATIENT.ID", VAL=NUM .
val_phrase(_,[id,"#",NUM|T],REL_ATT,VAL) :-
    name(NUM), REL_ATT="PATIENT.ID", VAL=NUM .
val_phrase("physician$'s",[N|T],REL_ATT,VAL) :-
    name(N), REL_ATT="PATIENT.LNAME", concat_quote(N,VAL) .
val_phrase("doctor$'s",[N|T],REL_ATT,VAL) :-
    name(N), REL_ATT="PATIENT.LNAME", concat_quote(N,VAL) .

concat_quote(N,VAL) :-
    concatenate("$'",N,X), concatenate(X,"$'",VAL) .

special("#") .

article(a) .
article(an) .
article(the) :- !.

adjective(next) .
adjective(any) .
adjective(social) .
adjective(any) .
adjective(all) .
adjective(temporary) .
adjective(archived) .
adjective(marital) :- !.
```

```
noun(patient) .
noun(patients) .
noun(doctor) .
noun(doctors) .
noun(location) .
noun(status) .
noun(condition) .
noun(examination) .
noun(date) .
noun(admission) .
noun(diagnosis) .
noun(computer) .
noun(information) .
noun(dr) .
noun(firstname) .
noun(lastname) .
noun("SIN") .
noun(insurance) .
noun(number) .
noun(no) .
noun("Dr") .
noun(physician) .
noun(address) .
noun(kin) .
noun(profession) .
noun(creation) .
noun(opening) .
noun(birth) .
noun(sex) .
noun(exam) .
noun(occupation) .
noun(birthdate) .
noun(id) .
noun(identification) .
noun(folder) .
noun(report) .
noun(reference) .
noun(names) .
noun(name) .
```

```
noun("identification#") .
noun("id#") .
noun("#") .
noun("doctor$'s") .
noun("physician$'s") .
noun(reports) .
noun(examinations) .
noun(room) :- !.


pronoun(who) .
pronoun(where) .
pronoun(when) .
pronoun(what) :- !.

verb_transitive(show) .
verb_transitive(get) .
verb_transitive(give) .
verb_transitive(list) .
verb_transitive(put) .
verb_transitive(tell) .
verb_transitive(display) :- !.

verb_copulative(is) .
verb_copulative(are) :- !.

auxiliary(has) :- !.

toobj(me) .
toobj(us) :- !.

preposition(of) .
preposition(in) .
preposition(with) .
preposition(about) .
preposition(on) :- !.
```

```
comp1(patient,["LNAME","FNAME"],"PATIENT",[]).
comp1(patients,["LNAME","FNAME"],"PATIENT",[]).
comp1(doctor,["DOCTNAME"],"PATIENT",[]).
comp1(doctors,["DOCTNAME"],"PATIENT",[]).
comp1(location,["PLOC"],"PATIENT",[]).
comp1(status,["STAT"],"PATIENT","DIAGNOSIS").
comp1(condition,["STAT"],"PATIENT","DIAGNOSIS").
comp1(diagnosis,["DIAG"],"PATIENT","DIAGNOSIS").
comp1(information,[*],"PATIENT","DIAGNOSIS").
comp1(firstname,["FNAME"],"PATIENT",[]).
comp1(lastname,["LNAME"],"PATIENT",[]).
comp1("SIN",["SIN"],"PATIENT",[]).
comp1(physician,["DOCTNAME"],"PATIENT",[]).
comp1(address,["PADD"],"PATIENT",[]).
comp1(profession,["PROF"],"PATIENT",[]).
comp1(occupation,["PROF"],"PATIENT",[]).
comp1(sex,["SEX"],"PATIENT",[]).
comp1(room,["PLOC"],"PATIENT",[]).
comp1(birthdate,["BDATE"],"PATIENT",[]).
comp1(id,["ID"],"PATIENT",[]).
comp1(identification,["PID"],"PATIENT",[]).
comp1("id#",["PID"],"PATIENT",[]).
comp1("identification#",["PID"],"PATIENT",[]).
comp1(examinations,["REPREF","EXAM"],"PATIENT","DIAGNOSIS").
comp1(examination,["REPREF","EXAM"],"PATIENT","DIAGNOSIS").
comp1(reports,["REPREF","EXAM"],"PATIENT","DIAGNOSIS").
comp1(report,["REPREF","EXAM"],"PATIENT","DIAGNOSIS").

comp2(exam,date,["EDATE"],"PATIENT","DIAGNOSIS").
comp2(examination,date,["EDATE"],"PATIENT","DIAGNOSIS").
comp2(admission,date,["ODATE"],"PATIENT",[]).
comp2(opening,date,["ODATE"],"PATIENT",[]).
comp2(creation,date,["CDATE"],"PATIENT",[]).
comp2(birth,date,["BDATE"],"PATIENT",[]).
comp2(marital,status,["MSTATUS"],"PATIENT",[]).
comp2(room,no,["PLOC"],"PATIENT",[]).
comp2(room,number,["PLOC"],"PATIENT",[]).
comp2(folder,status,["FS"],"PATIENT",[]).
comp2(patient,location,["PLOC"],"PATIENT",[]).
```

```
comp2(patient,identification,["PID"],"PATIENT",[]).
comp2(patient,id,["PID"],"PATIENT",[]).
comp2(report,reference,["REPREF"],"PATIENT","DIAGNOSIS").
comp2(date,birth,["BDATE"],"PATIENT",[]).
comp2(all,patients,["LNAME","FNAME"],"PATIENT",[]).
comp2(any,patient,["LNAME","FNAME"],"PATIENT",[]).
comp2(patient,names,["LNAME","FNAME"],"PATIENT",[]).
comp2(id,no,["PID"],"PATIENT",[]).
comp2(identification,no,["PID"],"PATIENT",[]).
comp2(id,"#",["PID"],"PATIENT",[]).
comp2(identification,number,["PID"],"PATIENT",[]).
comp2(identification,number,["PID"],"PATIENT",[]).
comp2(identification,"#",["PID"],"PATIENT",[]).
comp2("doctor$'s",name,["DOCTNAME"],"PATIENT",[]).
comp2("physician$'s",name,["DOCTNAME"],"PATIENT",[]).
comp2(physician,name,["DOCTNAME"],"PATIENT",[]).
comp2(doctor,name,["DOCTNAME"],"PATIENT",[]).
comp2(examination,number,["EXAM"],"PATIENT","DIAGNOSIS").
comp2(exam,number,["EXAM"],"PATIENT","DIAGNOSIS").
comp2(examination,no,["EXAM"],"PATIENT","DIAGNOSIS").
comp2(exam,no,["EXAM"],"PATIENT","DIAGNOSIS").
comp2(patient,status,["STAT"],"PATIENT","DIAGNOSIS").
comp2(patient,condition,["STAT"],"PATIENT","DIAGNOSIS").
comp2(archived,reports,["REPREF","EXAM"],"PATIENT","DIAGNOSIS").
comp2(temporary,reports,["REPREF","EXAM"],"PATIENT","DIAGNOSIS").
comp2(examination,reports,["REPREF","EXAM"],"PATIENT","DIAGNOSIS").
comp2(exam,reports,["REPREF","EXAM"],"PATIENT","DIAGNOSIS").

comp3(social,insurance,number,["SIN"],"PATIENT",[]).
comp3(next,exam,date,["NDATE"],"PATIENT","DIAGNOSIS").
comp3(social,insurance,no,["SIN"],"PATIENT",[]).
comp3(next,examination,date,["NDATE"],"PATIENT","DIAGNOSIS").
comp3(next,kin,address,["NOKADD"],"PATIENT",[]).

endmod /* parser */ .
```

Appendix B

/* This module executes the dynamic database queries and produces reults. It also checks for the error and accordingly reports to the MPROLOG MODULE. */

```
# include <stdio.h>
# include <ctype.h>

/* SQL stmt #1
EXEC SQL BEGIN DECLARE SECTION;
 VARCHAR pwd[10];
*/
struct {
 unsigned short len;
 unsigned char arr[10];
 } pwd;
/*
 VARCHAR uid[10];
*/
struct {
 unsigned short len;
 unsigned char arr[10];
 } uid;
/*
 VARCHAR stmt[256];
*/
/* SQL stmt #2
EXEC SQL END DECLARE SECTION;
*/


static struct {
unsigned int   sq001N;
unsigned char  *sq001V[4];
unsigned int   sq001L[4];
```

```
unsigned short  sq001T[4];
unsigned short *sq001I[4];
} sq001 = {4};
static struct {
unsigned int    sq002N;
unsigned char  *sq002V[1];
unsigned int    sq002L[1];
unsigned short  sq002T[1];
unsigned short *sq002I[1];
} sq002 = {1};
static struct {
unsigned int    sq003N;
unsigned char  *sq003V[1];
unsigned int    sq003L[1];
unsigned short  sq003T[1];
unsigned short *sq003I[1];
} sq003 = {1};
static struct {
unsigned int    sq004N;
unsigned char  *sq004V[1];
unsigned int    sq004L[1];
unsigned short  sq004T[1];
unsigned short *sq004I[1];
} sq004 = {1};
static struct {
unsigned int    sq005N;
unsigned char  *sq005V[1];
unsigned int    sq005L[1];
unsigned short  sq005T[1];
unsigned short *sq005I[1];
} sq005 = {1};
static struct {
unsigned int    sq006N;
unsigned char  *sq006V[1];
unsigned int    sq006L[1];
unsigned short  sq006T[1];
unsigned short *sq006I[1];
} sq006 = {1};
static struct {
```

```
unsigned int   sq007N;
unsigned char  *sq007V[1];
unsigned int   sq007L[1];
unsigned short  sq007T[1];
unsigned short *sq007I[1];
} sq007 = {1};
static struct {
unsigned int   sq008N;
unsigned char  *sq008V[1];
unsigned int   sq008L[1];
unsigned short  sq008T[1];
unsigned short *sq008I[1];
} sq008 = {1};
static struct {
unsigned int   sq009N;
unsigned char  *sq009V[1];
unsigned int   sq009L[1];
unsigned short  sq009T[1];
unsigned short *sq009I[1];
} sq009 = {1};
static struct {
unsigned int   sq010N;
unsigned char  *sq010V[1];
unsigned int   sq010L[1];
unsigned short  sq010T[1];
unsigned short *sqC10I[1];
} sq010 = {1};
static struct {
unsigned int   sq011N;
unsigned char  *sq011V[1];
unsigned int   sq011L[1];
unsigned short  sq011T[1];
unsigned short *sq011I[1];
} sq011 = {1};
static struct {
unsigned int   sq012N;
unsigned char  *sq012V[1];
unsigned int   sq012L[1];
unsigned short  sq012T[1];
```

```
unsigned short *sq012I[1];
} sq012 = {1};
static struct {
unsigned int   sq013N;
unsigned char  *sq013V[1];
unsigned int   sq013L[1];
unsigned short  sq013T[1];
unsigned short *sq013I[1];
} sq013 = {1};
static struct {
unsigned int   sq014N;
unsigned char  *sq014V[1];
unsigned int   sq014L[1];
unsigned short  sq014T[1];
unsigned short *sq014I[1];
} sq014 = {1};
static struct {
unsigned int   sq015N;
unsigned char  *sq015V[1];
unsigned int   sq015L[1];
unsigned short  sq015T[1];
unsigned short *sq015I[1];
} sq015 = {1};
static int SQLTM[8];
static int sqlcun[1] = {
0};
static int sqlusi[1] = {
0};
static unsigned int sqlami = 0;
static int SQLBT0 = 1;
static int SQLBT1 = 2;
static int SQLBT2 = 4;
static int SQLBT3 = 9;
static unsigned int sqlvsn = 10110;
extern   sqlad2();
extern   sqlbs2();
extern   sqlcls();
extern   sqlcom();
extern   sqlexe();
```

```
extern   sqlfch();
extern   sqlgb2();
extern   sqlgd2();
extern   sqllo2();
extern   sqlopn();
extern   sqlos2();
extern   sqlrol();
extern   sqlsca();
extern   sqlscc();
extern   sqlsch();
extern   sqltfl();
/* SQL stmt #3
EXEC SQL INCLUDE SQLCA;
*/
/*
NAME
 SQLCA : SQL Communications Area.
FUNCTION
 Contains no code. Oracle fills in the SQLCA with status info
 during the execution of a SQL stmt.

#ifndef SQLCA
#define SQLCA 1

struct  sqlca
        {
        /* ub1 */ char    sqlcaid[8];
        /* b4  */ long    sqlabc;
        /* b4  */ long    sqlcode;
        struct
          {
          /* ub2 */ unsigned short sqlerrml;
          /* ub1 */ char           sqlerrmc[70];
          } sqlerrm;
        /* ub1 */ char    sqlerrp[8];
        /* b4  */ long    sqlerrd[6];
        /* ub1 */ char    sqlwarn[8];
        /* ub1 */ char    sqlext[8];
        };
```

```
#ifdef  SQLCA_STORAGE_CLASS
SQLCA_STORAGE_CLASS struct sqlca sqlca
#else
      struct sqlca sqlca
#endif

#ifdef  SQLCA_INIT
      = {
         {'S', 'Q', 'L', 'C', 'A', ' ', ' ', ' '},
      sizeof(struct sqlca),
      0,
       { 0, {0}},
         {'N', 'O', 'T', ' ', 'S', 'E', 'T', ' '},
       {0, 0, 0, 0, 0, 0},
       {0, 0, 0, 0, 0, 0, 0, 0},
       {0, 0, 0, 0, 0, 0, 0, 0}
      }
#endif
      ;

#endif

/* end SQLCA */

/* SQL stmt #4
EXEC SQL INCLUDE SQLDA.H;
*/

#ifndef SQLDA_
#define SQLDA_ 1

#ifdef T
# undef T
#endif
#ifdef F
# undef F
#endif
```

```
struct SQLDA {
 int    N;  /* Descriptor size in number of entries        */
 char  **V;  /* Ptr to Arr of addresses of main variables   */
 int   *L;  /* Ptr to Arr of lengths of buffers            */
 short  *T;  /* Ptr to Arr of types of buffers             */
 short **I;  /* Ptr to Arr of addresses of indicator vars   */
 int    F;  /* Number of variables found by DESCRIBE       */
 char  **S;  /* Ptr to Arr of variable name pointers        */
 short  *M;  /* Ptr to Arr of max lengths of var. names     */
 short  *C;  /* Ptr to Arr of current lengths of var. names */
 char  **X;  /* Ptr to Arr of ind. var. name pointers       */
 short  *Y;  /* Ptr to Arr of max lengths of ind. var. names */
 short  *Z;  /* Ptr to Arr of cur lengths of ind. var. names */
 };

typedef struct SQLDA SQLDA;

#endif
SQLDA *bdp;
SQLDA *sdp;
short *sdt = 0;
int sdtl;
char *vars = 0;
int bdsize = 5;
int bvsize = 10;
int sdsize = 5;
int svsize = 80;
extern char *malloc();
extern char *sqlald();

register int min(x, y)

register int x, y;

{
     return((x <= y) ? x : y);
}

main()
```

```
{
 int i;
 FILE *fopen(), *fp;
 char line1[240],line2[80],line3[80];
 fp = fopen("query","r");
 strcpy(uid.arr,"goutam");
 uid.len = strlen(uid.arr);
 strcpy(pwd.arr,"goutam");
 pwd.len = strlen(pwd.arr);
/* SQL stmt #5
 EXEC SQL WHENEVER SQLERROR err;
 EXEC SQL CONNECT :uid IDENTIFIED BY :pwd;
*/
sqlsca(&sqlca);
sq001.sq001V[0] = (unsigned char *)&uid.len;
sq001.sq001L[0] = (unsigned int)12;
sq001.sq001T[0] = (unsigned short)9;
sq001.sq001I[0] = (unsigned short *)0;
sq001.sq001V[1] = (unsigned char *)&pwd.len;
sq001.sq001L[1] = (unsigned int)12;
sq001.sq001T[1] = (unsigned short)9;
sq001.sq001I[1] = (unsigned short *)0;
sq001.sq001T[2] = (unsigned short)10;
sq001.sq001T[3] = (unsigned short)10;
SQLTM[0] = (int)0;
SQLTM[1] = (int)10;
sqllo2(
  &sq001.sq001N,sq001.sq001V,sq001.sq001L,sq001.sq001T,
 &sqlami, &SQLTM[0], &SQLTM[1], &sqlvsn);
if (sqlca.sqlcode < 0) exit(0x1);
 printf("Welcome to the dynamic SQL \n");
 if (sqlca.sqlcode != 0) goto errrpt;
 bdp = (SQLDA *)sqlald(bdsize,bvsize,10);
 if (bdp == NULL)
  { puts("ALLOCATION FAIL :bdp = sqlald(bdsize,bvsize,10)");
    exit(1);
  }
 sdp = (SQLDA *)sqlald(sdsize,svsize,0);
 if (sdp == NULL)
```

```
 { puts("ALLOCATION FAIL : sdp = sqlald(sdsize,svsize,0)");
   { printf("cannot allocate memory for sdp \n"); exit(1);
  }
 sdp->N = 0;
 fscanf(fp,"%s\n%s\n%s\n",line1,line2,line3);
 strcat(line1,line2);
 strcat(line1,line3);
/* SQL stmt #7
 EXEC SQL PREPARE S FROM :line1;
*/
sqlsca(&sqlca);
if ( !sqlusi[0] )
  {  /* OPEN SCOPE */
sq002.sq002T[0] = (unsigned short)10;
SQLTM[0] = (int)4;
sqlbs2(&sq002.sq002N, sq002.sq002V,
   sq002.sq002L, sq002.sq002T, sq002.sq002I,
  &SQLTM[0], &sqlusi[0]);
  }  /* CLOSE SCOPE */
sqlsch(&sqlusi[0]);
sqlscc(&sqlcun[0]);
sqltfl(&SQLTM[0], &SQLBT0);
if ( !SQLTM[0] )
  {  /* OPEN SCOPE */
SQLTM[0] = (int)16384;
sqlopn(&SQLTM[0], &SQLBT0, &sqlvsn);
  }  /* CLOSE SCOPE */
sq003.sq003V[0] = (unsigned char *)&stmt.len;
sq003.sq003L[0] = (unsigned int)258;
sq003.sq003T[0] = (unsigned short)9;
sq003.sq003I[0] = (unsigned short *)0;
sqlos2(&sq003.sq003N, sq003.sq003V,
   sq003.sq003L, sq003.sq003T, sq003.sq003I);
if (sqlca.sqlcode < 0) exit(0x1);
 /* if (sqlca.sqlwarn[0] == 'W') mufwrn(); */
 /* printf("prepare ok\n"); */
/* SQL stmt #8
 EXEC SQL DECLARE C CURSOR FOR S;
 */
```

```
 /* printf("cursor ok\n"); */
 bdp->N = bdsize;
/* SQL stmt #9
 EXEC SQL DESCRIBE BIND VARIABLES FOR S INTO bdp;
*/
sqlsca(&sqlca);
if ( !sqlusi[0] )
  { /* OPEN SCOPE */
sq004.sq004T[0] = (unsigned short)10;
SQLTM[0] = (int)4;
sqlbs2(&sq004.sq004N, sq004.sq004V,
  sq004.sq004L, sq004.sq004T, sq004.sq004I,
 &SQLTM[0], &sqlusi[0]);
  } /* CLOSE SCOPE */
sqlsch(&sqlusi[0]);
sqlscc(&sqlcun[0]);
sqlgb2(&bdp->N,&bdp->F,bdp->S,bdp->M,bdp->C,
  bdp->X,bdp->Y,bdp->Z,bdp->T);
if (sqlca.sqlcode < 0) exit(0x1);
 /* printf("%2d",bdp->N);printf("%2d\n",bdp->F); */
 if (bdp->F < 0)
   {
   bdsize = -(bdp->F);
   sqlclu(bdp);
   bdp = (SQLDA *)sqlald(bdsize,10,0);
/* SQL stmt #10
   EXEC SQL DESCRIBE BIND VARIABLES FOR S INTO bdp;
*/
sqlsca(&sqlca);
if ( !sqlusi[0] )
  { /* OPEN SCOPE */
sq005.sq005T[0] = (unsigned short)10;
SQLTM[0] = (int)4;
sqlbs2(&sq005.sq005N, sq005.sq005V,
  sq005.sq005L, sq005.sq005T, sq005.sq005I,
 &SQLTM[0], &sqlusi[0]);
  } /* CLOSE SCOPE */
sqlsch(&sqlusi[0]);
sqlscc(&sqlcun[0]);
```

```
sqlgb2(&bdp->N,&bdp->F,bdp->S,bdp->M,bdp->C,
  bdp->X,bdp->Y,bdp->Z,bdp->T);
if (sqlca.sqlcode < 0) exit(0x1);
   }
 bdp->N = bdp->F;
/* SQL stmt #11
 EXEC SQL OPEN;
*/
sqlsca(&sqlca);
if ( !sqlusi[0] )
  { /* OPEN SCOPE */
sq006.sq006T[0] = (unsigned short)10;
SQLTM[0] = (int)4;
sqlbs2(&sq006.sq006N, sq006.sq006V,
   sq006.sq006L, sq006.sq006T, sq006.sq006I,
  &SQLTM[0], &sqlusi[0]);
  } /* CLOSE SCOPE */
sqlsch(&sqlusi[0]);
sqlscc(&sqlcun[0]);
SQLTM[0] = (int)1;
sqlexe(&SQLTM[0]);
if (sqlca.sqlcode < 0) exit(0x1);
 /* printf("openc using desc bdp ok\n");  */
 sdp->N = sdsize;
/* SQL stmt #12
 EXEC SQL DESCRIBE SELECT LIST FOR S INTO sdp;
*/
 sqlsca(&sqlca);
 if ( !sqlusi[0] )
   { /* OPEN SCOPE */
 sq007.sq007T[0] = (unsigned short)10;
 SQLTM[0] = (int)4;
 sqlbs2(&sq007.sq007N, sq007.sq007V,
    sq007.sq007L, sq007.sq007T, sq007.sq007I,
   &SQLTM[0], &sqlusi[0]);
   } /* CLOSE SCOPE */
 sqlsch(&sqlusi[0]);
 sqlscc(&sqlcun[0]);
 sqlgd2(&sdp->N, &sdp->F, sdp->S, sdp->M,
```

```
  sdp->C, sdp->L, sdp->T);
if (sqlca.sqlcode < 0) exit(0x1);
 /* printf("%3d",sdp->N);printf("%3d\n",sdp->F); */
 if (sdp->F < 0)
    {
    sdsize = -(sdp->F);
    sqlclu(sdp);
    sdp = (SQLDA *)sqlald(sdsize,10,0);
/* SQL stmt #13
   EXEC SQL DESCRIBE SELECT LIST FOR S INTO sdp;
*/
sqlsca(&sqlca);
if ( !sqlusi[0] )
  { /* OPEN SCOPE */
sq008.sq008T[0] = (unsigned short)10;
SQLTM[0] = (int)4;
sqlbs2(&sq008.sq008N, sq008.sq008V,
  sq008.sq008L, sq008.sq008T, sq008.sq008I,
 &SQLTM[0], &sqlusi[0]);
  } /* CLOSE SCOPE */
sqlsch(&sqlusi[0]);
sqlscc(&sqlcun[0]);
sqlgd2(&sdp->N, &sdp->F, sdp->S, sdp->M,
  sdp->C, sdp->L, sdp->T);
if (sqlca.sqlcode < 0) exit(0x1);
    }
 /* printf("before N changes\n");  */
 sdp->N = sdp->F;
 /* printf("%3d\n",sdp->N); */
 if (sdp->F != 0) fillSelDesc();
 printf("fillSelDesc ok\n");
 if (sdp->N != 0) doFetches();
 else
  printf("%u rows processed.\n",sqlca.sqlerrd[2]);
/* SQL stmt #14
 EXEC SQL CLOSE C;
*/
sqlsca(&sqlca);
if ( !sqlusi[0] )
```

```
 {  /* OPEN SCOPE */
sq009.sq009T[0] = (unsigned short)10;
SQLTM[0] = (int)4;
sqlbs2(&sq009.sq009N, sq009.sq009V,
  sq009.sq009L, sq009.sq009T, sq009.sq009I,
 &SQLTM[0], &sqlusi[0]);
 }  /* CLOSE SCOPE */
sqlsch(&sqlusi[0]);
sqlscc(&sqlcun[0]);
SQLTM[0] = (int)1;
sqlcls(&SQLTM[0]);
if (sqlca.sqlcode < 0) exit(0x1);

/* SQL stmt #15
 EXEC SQL COMMIT WORK RELEASE;
*/
sqlsca(&sqlca);
if ( !sqlusi[0] )
  {  /* OPEN SCOPE */
sq010.sq010T[0] = (unsigned short)10;
SQLTM[0] = (int)4;
sqlbs2(&sq010.sq010N, sq010.sq010V,
  sq010.sq010L, sq010.sq010T, sq010.sq010I,
 &SQLTM[0], &sqlusi[0]);
  }  /* CLOSE SCOPE */
sqlsch(&sqlusi[0]);
SQLTM[0] = (int)1;
sqlcom(&SQLTM[0]);
if (sqlca.sqlcode < 0) exit(0x1);
/* SQL stmt #16
 EXEC SQL ROLLBACK WORK RELEASE;
*/
sqlsca(&sqlca);
if ( !sqlusi[0] )
  {  /* OPEN SCOPE */
sq011.sq011T[0] = (unsigned short)10;
SQLTM[0] = (int)4;
sqlbs2(&sq011.sq011N, sq011.sq011V,
  sq011.sq011L, sq011.sq011T, sq011.sq011I,
```

```
  &SQLTM[0], &sqlusi[0]);
  } /* CLOSE SCOPE */
sqlsch(&sqlusi[0]);
SQLTM[0] = (int)1;
sqlrol(&SQLTM[0]);
if (sqlca.sqlcode < 0) exit(0x1);
 return;

 err:
      printf("\n%.70s \n",sqlca.sqlerrm.sqlerrmc);
 fclose(fp);
/* SQL stmt #17
     EXEC SQL CLOSE C;
*/
sqlsca(&sqlca);
if ( !sqlusi[0] )
  { /* OPEN SCOPE */
sq012.sq012T[0] = (unsigned short)10;
SQLTM[0] = (int)4;
sqlbs2(&sq012.sq012N, sq012.sq012V,
  sq012.sq012L, sq012.sq012T, sq012.sq012I,
  &SQLTM[0], &sqlusi[0]);
  } /* CLOSE SCOPE */
sqlsch(&sqlusi[0]);
sqlscc(&sqlcun[0]);
SQLTM[0] = (int)1;
sqlcls(&SQLTM[0]);
if (sqlca.sqlcode < 0) exit(0x1);
/* SQL stmt #18
     EXEC SQL COMMIT WORK RELEASE;
*/
sqlsca(&sqlca);
if ( !sqlusi[0] )
  { /* OPEN SCOPE */
sq013.sq013T[0] = (unsigned short)10;
SQLTM[0] = (int)4;
sqlbs2(&sq013.sq013N, sq013.sq013V,
  sq013.sq013L, sq013.sq013T, sq013.sq013I,
  &SQLTM[0], &sqlusi[0]);
```

```
 } /* CLOSE SCOPE */
sqlsch(&sqlusi[0]);
SQLTM[0] = (int)1;
sqlcom(&SQLTM[0]);
if (sqlca.sqlcode < 0) exit(0x1);
/* SQL stmt #19
     EXEC SQL ROLLBACK WORK RELEASE;
*/
sqlsca(&sqlca);
if ( !sqlusi[0] )
 { /* OPEN SCOPE */
sq014.sq014T[0] = (unsigned short)10;
SQLTM[0] = (int)4;
sqlbs2(&sq014.sq014N, sq014.sq014V,
  sq014.sq014L, sq014.sq014T, sq014.sq014I,
 &SQLTM[0], &sqlusi[0]);
 } /* CLOSE SCOPE */
sqlsch(&sqlusi[0]);
SQLTM[0] = (int)1;
sqlrol(&SQLTM[0]);
if (sqlca.sqlcode < 0) exit(0x1);
 fclose(fp);
     return;
/* end main() */
}

/* EXEC SQL WHENEVER SQLERROR STOP;
EXEC SQL WHENEVER SQL WARNING CONTINUE;
EXEC SQL WHENEVER NOT FOUND CONTINUE; */


doFetches()
{
 int cnt;
 char colname[100];
 int colnamel;
/* SQL stmt #20
 EXEC SQL WHENEVER NOT FOUND GOTO not_found;
*/
```

```
 putchar('\n');
 for (cnt = 0;cnt<sdp->N;cnt++)
  {
   colnamel = min(sdp->L[cnt],sdp->C[cnt]);
   colnamel = min(colnamel,sizeof(colname)-1);
   memcpy(colname,sdp->S[cnt],colnamel);
   colname[colnamel] = '\0';

   if (sdt[cnt] == 2 )
    printf("%*s ",sdp->L[cnt],colname);
   else printf("%-*s",sdp->L[cnt],colname);
  }
 putchar('\n');
 for (cnt = 0; ; cnt++)
  {
/* SQL stmt #21
   EXEC SQL FETCH C USING DESCRIPTOR sdp ;
*/
sqlsca(&sqlca);
if ( !sqlusi[0] )
 { /* OPEN SCOPE */
sq015.sq015T[0] = (unsigned short)10;
SQLTM[0] = (int)4;
sqlbs2(&sq015.sq015N, sq015.sq015V,
 sq015.sq015L, sq015.sq015T, sq015.sq015I,
 &SQLTM[0], &sqlusi[0]);
 } /* CLOSE SCOPE */
sqlsch(&sqlusi[0]);
sqlscc(&sqlcun[0]);
sqltfl(&SQLTM[0], &SQLBT2);
if ( !SQLTM[0] )
 { /* OPEN SCOPE */
sqlad2(&sdp->N, sdp->V,
 sdp->L, sdp->T, sdp->I);
 } /* CLOSE SCOPE */
SQLTM[0] = (int)1;
SQLTM[1] = (int)0;
sqlfch(&SQLTM[0], &SQLTM[1]);
if (sqlca.sqlcode == 1403)  goto not_found;
```

```
if (sqlca.sqlcode < 0) exit(0x1);
   prnt(sdp);
  }
 not_found: printf("\n%u row(s) selected\n",cnt);
/* SQL stmt #22
 EXEC SQL WHENEVER NOT FOUND CONTINUE;
*/
 /*  end doFetches */
}

prnt(dp)
SQLDA *dp;
{
 int i;
 int *vp;
 short *ip;
 for (i = 0; i < dp->N ; i++)
   {
   /* ip = dp->I[i];
   if(*ip < 0)
    printf("%-*s",dp->L[i]," ");
   else */
    printf("%-.*s",dp->L[i],dp->V[i]);
   }
 putchar('\n');
}
/* preparing the select list */
fillSelDesc()
{
 int i;
 unsigned char prec;
 char scale;
 if ( !sdt )
   { sdt = (short *)calloc(sdp->N,sizeof(short) );
    /* printf("%3d\n",sdtl); */
   }
 else if ( sdtl < sdp->N )
   { sdt = (short *)realloc(sdt,sizeof(short) * sdp->N);
   }
```

```
 sdtl = sdp->N; /* printf("%3d\n",sdtl); */
 for (i = 0; i < sdp->N; i++)
   { /* printf("%3d\n",sdp->T[i]); */
   /* sdp->T[i] = (sdp->T[i] & 0x8000); */
   sdt[i] = sdp->T[i];
   if ( sdp->T[i] == 2 )
    { prec = (unsigned char)(sdp->L[i] >> 8);
     scale = (char)sdp->L[i];
     if (prec == 0 ) prec = 26;
     sdp->L[i] = prec;
     if (scale < 0 ) sdp->L[i] += -scale;
     sdp->L[i] += 2;
    }
   else if (sdp->T[i] == 12 ) { sdp->L[i] = 9; }
   sdp->T[i] = 1;
   sdp->L[i] = min(sdp->L[i],240);
   sdp->V[i] = malloc(sdp->L[i]);
   /* printf("%3d\n",sdt[i]); */
   /* sdp->I[i] = malloc(sizeof(short) ); */
   }
/* end of fillSelDesc */
}
```