RULES FOR MAPPING A CONCEPTUAL MODEL ONTO VARIOUS DATA BASE MANAGEMENT SYSTEMS.

Volume 1

A thesis submitted for the degree of Doctor of Philosophy

by

Jayasri Chaudhuri

University of Aston in Birmingham

February 1988

RULES FOR MAPPING A CONCEPTUAL MODEL ONTO VARIOUS DATA BASE MANAGEMENT SYSTEMS.

A thesis submitted for the degree of Doctor of Philosophy

by

Jayasri Chaudhuri

University of Aston in Birmingham

February 1988

ABSTRACT

The design and implementation of data bases involve, firstly, the formulation of a conceptual data model by systematic analysis of the structure and information requirements of the organisation for which the system is being designed; secondly, the logical mapping of this conceptual model onto the data structure of the target data base management system (DBMS); and thirdly, the physical mapping of this structured model into storage structures of the target DBMS. The accuracy of both the logical and physical mapping determine the performance of the resulting system.

This thesis describes research which develops software tools to facilitate the implementation of data bases. A conceptual model describing the information structure of a hospital is derived using the Entity-Relationship (E-R) approach and this model forms the basis for mapping onto the logical model. Rules are derived for automatically mapping the conceptual model onto relational and CODASYL types of data structures. Further algorithms are developed for partly automating the implementation of these models onto INGRES, MIMER and VAX-11 DBMS.

KEYWORDS

Conceptual Modelling

Data Bases

Logical Mapping

Physical Mapping

ACKNOWLEDGEMENTS

The author would like to express her gratitude to the following people:

Mr. D.E. Avison, BA, MSc, FBCS, her supervisor, for his guidance, help, and patience throughout the duration of this research project.

Staff of George Eliot Hospital, Nuneaton, for their cooperation during the data analysis phase.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 Introduction

An information system provides storage of data (data base) and a flow of information to satisfy the requirements of users. Within an organisation the requirements of different groups of users are varied, although these different groups of users may share common data. Problems have been caused by unnecessary duplication of data, a lack of consistency of data and a lack of flexibility in updating application programs. Data Base Management Systems (DBMS) address these problems by managing the data base. We need to establish a more rigorous definition for information systems, data bases and DBMS before discussing the features of DBMS.

Information systems: Davis and Olson (1985, pp 6) describe an information system as:

> "...an integrated, user machine system for providing information to support operations, management, analysis and decision-making functions in an organisation. The system utilises computer hardware and software; manual procedures; models for analysis, planning, control, and decision making; and a data base."

Information should be provided to the recipient at the right time, and in a form that is understandable, accurate, and relevant to the users' needs. Users should agree to the objectives and features of the information system before implementation. In order to achieve these objectives it is necessary to perform a systematic analysis of the structure and information requirements of the organisation for which the system is being designed. Such an analysis yields, amongst other things, the conceptual model which provides the framework on which a computer based information system may be developed. Conceptual modelling is discussed in some detail in section 1.3.1.

Data bases and DBMS: Date (1986) and Kroenke (1983) both perceive a data base as a collection of operational data used by the application systems of some particular organisation. A data base provides the organisation with centralised control of its data. A DBMS is a software tool for manipulating a data base. Martin (1985, pp 4) distinguishes between a DBMS and a conventional system of computer files as follows:

> "The most important difference between a data-base management system and a file management system is that data-base management translates between the application program's view of data and the actual structure of the data. It preserves the program's view of data when the actual view changes in either a logical or a physical manner."

Martin is making the important distinction here between logical (the perceived structure of the data) and physical (the way in which the data is actually stored) data structures which provide data independence.

It is argued by Avison (1985), Date (1986), and many others that the use of DBMS is justified because they aim to be flexible and cost effective. Once the data is captured it can be used by a number of applications. Even if the requirements change, there is usually only a need to modify the application programs. It is not usually necessary to change the system completely because the data is still likely to be relevant to the new circumstance. Fry and Deppe (1976) and Fry and Sibley (1976), also, point out that one of the reasons for using DBMS is that they greatly reduce maintenance effort due to the data independence provided.

This research concerns automation of some aspects of the data base design process. Therefore, the architecture of a DBMS and data base design are discussed in detail.

1.2 DBMS architecture

Whilst it is generally accepted that a multiple-level architecture is needed for a DBMS to achieve a high degree of data independence, there are still differences of opinion as to the number of levels of abstraction necessary.

Kerschberg et al (1976) recognise the need for a multiple-level DBMS architecture but do not propose any definite architecture.

Senko et al (1973) describe a DBMS architecture in terms of four levels of abstraction.

However, this approach and those of many others, for example CODASYL (1971), have not been widely adopted but they are not too dissimilar to the most widely accepted architecture proposed by the ANSI/X3/SPARC (1977,1978) committee. The latter consists of three levels of abstraction: external, logical and internal.

<u>A logical schema</u>	This is a set of rules describing which data may enter and reside in the data base described as the logical model in this thesis.
<u>An internal schema</u>	This is a set of rules describing how the data is physically represented on storage media described as the physical model in this thesis.
<u>An external schema</u>	This is a set of rules describing how data is viewed by an application program described as the external model in this thesis.

The three-level architecture has been adopted in this research because it has been the most widely accepted, and most commercially available DBMS are based on this approach. This design prevents changes made to one part of the data base affecting any other part of that data base. If an external model changes, then only the mapping between the external model and the logical model needs to be changed. Similarly, changes in an internal model only require changes in the mapping between the internal and logical models. The conceptual model serves as the reference point for the mappings, hence the stress paid on the conceptual modelling in this research.

The success of a three-level architecture depends on how well the user requirements have been analysed to form the conceptual model and the accuracy of the mapping to the logical, physical and external models of the DBMS.

1.3 Data base design

This is the process of determining the content and the arrangement of data in the data base that is required to fulfil the functional requirements of the users and to provide an acceptable level of performance. A good data base design process should produce a data base which meets the objectives of the users. Further, this process should be usable by designers with different degrees of experience in data base design. Novak and Fry (1976) argue that in order to be able to achieve these objectives the design process should be broken down into phases consisting of simpler tasks. The result produced in each phase should be used in the following phase until a suitable structure for implementation is achieved.

Kahn (1976) describes two types of data available to a data base designer. The first type of data is the Information Structure Perspective (ISP). This is the natural structure of the data and is not bound to any application. The second type is the Usage Perspective (UP) which describes the data by the way in which it is used by various applications. Data base design based on ISP gives a flexible design which supports various unstructured applications, but it does not, necessarily, provide an efficient data base. Data base design based on UP data produces a tailor-made data base which supports the anticipated applications efficiently because the design is based on the most used access paths. ISP provides flexibility but UP provides efficiency for a limited number of applications.

By using both perspectives, a better design (flexible and efficient) will result. Kahn (1978) and Novak and Fry (1976) suggest that ISP should be used for initial design and UP should be used to refine the design. Atre (1980), Chen (1977), DeMarco (1978), Finkelstein (1980) and King (1977) give support to the view that both perspectives should be adopted. They do not specifically discuss ISP and UP, but they all argue that in the first stage of data base design the data resources of the enterprise should be modelled independently of how the data is accessed or used. Furthermore, this must be achieved independently of the DBMS used.

However, whilst these authors agree on the principle that the data resources should be modelled, they adopt different approaches to the modelling process. For example, DeMarco takes a functional approach whereas Chen follows a data approach.

The data base design process used in this project is based on the two-level view (ISP and UP) described above and consists of the following three phases:

Conceptual modelling	the first phase, which is the process of identifying the data resources of the enterprise and organising them to form a structured data model, known as the conceptual model.
Logical mapping	the second phase, which is the process of transforming the conceptual model on to the data structure of the logical model of the DBMS on which the data base is going to be implemented.
Physical mapping	the third phase, which is the process of mapping the data on to the storage and access structure of the target DBMS, depending on the usage pattern of the data.

This research focuses on ways of automating the logical and physical mapping process.

1.3.1 Conceptual modelling

The ANSI-SPARC (1977) report defines the conceptual model as 'a description of an "enterprise", that is, such a part of an organisation which is of interest to the community of persons wanting to communicate with the data base'. The conceptual model includes a description of the object types (entities) important to the organisation, the relationships existing between them, their properties (or attributes), the specification of the constraints and the semantic information that defines the valid states of the data. This is used as a framework around which the organisation's information system is developed.

Many proposals for a conceptual model have been put forward.

The ANSI-SPARC report argues that a conceptual model is necessary but does not specify the shape of that model. Some of the important issues that should be satisfied are evolvability (changes in the real world should only involve a few modifications to the conceptual model), transformability (the mapping between the conceptual model and physical model or external model should be as simple as possible) and stability (changes in the external or internal model should not affect the conceptual model).

Information algebra (see CODASYL Development Committee 1962), one of the earlier models proposed, contains two important concepts: the concept of an entity as an object that has reality and the concept of joining records on equal values of record identifiers (keys). When attempting to model reality, it is essential to distinguish between different objects in the real world and understand how they are related to one another. This model used a complex hierarchical structure which was unstable to changes in reality. It also failed to make a clear distinction between the conceptual level and the data processing level. Langefors (1974) tried to solve the problems arising in putting linguistic references to objects in the real world by introducing the concept of elementary messages which were binary 'entity-property' relationships. Each message described only one property of an object. There was no attempt made to distinguish between properties of objects and relationships between objects.

The information structure in Mealy (1976) consists of binary 'data-maps' between entities and properties and binary 'structured-maps' between entities and entities. The concept of 'data-maps' is equivalent to Langefors elementary messages. But Mealy's model distinguishes between properties and relationships and thus the concept of 'structured-maps' has been introduced to show the relationship between two entities.

Network, hierarchical and relational structures have also been used for conceptual modelling. Tsichritzis and Lochovsky (1982) describe a network data model as a set of records and a set of links. Each link type consists of a parent record type and a child record type. Each occurrence of a link has a single occurrence of the parent record and a set of child records. A hierarchical data model consists of a tree structure. A tree consists of a root record type and an ordered set of dependent subtree types. The tree consists of a hierarchical arrangement of record types.

From the definitions of these structures, it is obvious that networks and hierarchies both have access path dependence built into their structure due to the restricted hierarchical arrangement of the data structure. The data description methods are often so complex that they cannot be used to describe a conceptual model.

The relational model, described by Codd (1970), has only one basic construct, the relation. A relation is a two dimensional table. A row corresponds to an object and is called a tuple. The columns correspond to the properties of the objects and are called attributes. The degree of a relation is equal to the number of columns in the relation. A binary relation consists of two columns, whereas a n-ary relation consists of 'n' columns. This tabular representation of data relationships is not dependent on any specific physical implementation and, therefore proves to be a good choice as a descriptive medium for the conceptual model.

Schmid (1977) points out that there are some semantic difficulties with the pure n-ary relational model. The relational model does not give any indication about the way in which the relations are associated with each other. His model is represented by objects (entities or entity names), object types (entity sets), irreducible relations (relations which cannot be broken down any further) called associations, and characteristics (descriptions of objects). The division of associations and characteristics is the same as the separation of relationships and properties.

Thus, for example, 'employee' is an object type, 'employee-car' is an association, whereas, 'employee-weight' is a characteristic. Schmid argues that the distinction helps in formulating consistency rules. For example the characteristic 'employee-weight' has no significance without the object type 'employee' whereas, because there is an association defined over 'employee' and 'car', a 'car' can exist independently without being associated with any other object type. A number of data models have been developed using this concept, for example, those proposed by Benci et al (1976) and Moulin et al (1977).

Hall et al (1976) argue that the best conceptual model would be formed by combining the characteristics of the relational model with the entity model. Their model consists of entity sets, value sets which are sets of values from which certain objects and their properties can draw their actual values, and irreducible relations. For example, 'employee' is an entity set and the set of integers from 18 to 65 is a value set from which the property 'employee-age' can take its value. Hall et al do not advocate the invention of a set of unique names for each entity set, but they do realise that any set of attributes will not give unique identification. Thus their model includes entity surrogates within a relational framework. Surrogates are collections of unique objects which act as the representatives of the objects in the outside world. To refer to an object in the model, we have to refer to its surrogate using one or more properties to uniquely identify the surrogate required. The major innovation of this model is its recognition that n-aries should be reduced to the size of basic facts. Hall et al put forward a mathematical process which will reduce n-ary relations to basic facts. They believe that irreducible n-aries have advantages over pure binaries because information about an object type can be represented by a single n-ary relation, whereas it will take more than one binary relation to represent all the properties of an object type.

Bracchi et al (1976), on the other hand, argue that n-ary relations contain a number of basic facts mixed up in an indistinguishable manner and thus do not represent the semantics of the real world. One crucial requirement of a conceptual model is stability, that is, the characteristic of being unaffected by changes in the internal or external model. To ensure stability, they argue that the conceptual model should be made up of small unstructured concepts. Their model is based on binary logical associations. The structures of the binary association do not follow from functional dependencies. Functional dependency defines the functional mapping between two entities, that is, when the value of the first entity is known, the value of the other entity can be determined from that value. Not all relations in the real world are binary in nature. Non-binary facts are represented by internal sets of concepts which have no meaning but link together two other concepts. This introduces many pseudo entities in the model.

In the Entity-Relationship (E-R) model of Chen (1976), real world information is represented by entities and relationships. This is a unification of the views held by the relational, network and entity set model. In his model, objects of interest are known as entities which can be classified into entity sets having the same properties. Entities are associated with each other by a relationship. Attributes define the properties of entities as well as relationships. This model does not allow the representation of relationships between two relationships or between an entity and a relationship. Howe (1983), Robinson (1985) and many others argue that the semantics of data and functional dependencies are better represented in this model. The graphical notation consisting of boxes and arrows used in the E-R model was first developed by Bachman (1969). Senko et al (1973) attempted to formalise this notation and produced their entity set model. Approaches developed by Bachman and Daya (1977) and Palmer (1978), which were proposed after Chen, are modifications of this E-R approach. View integration, that is integrating all user views to form a consistent global view, is part of Chen's conceptual design process. Other methods proposed by Navathe and Schkolnick (1978), Yao (1977) and Yao and De Jong (1978) also consider view integration as a part of the conceptual design. Lum (1979), on the other hand, advocates that view integration should be carried out later.

Falkenberg (1976) argues that the distinction between relationships and properties on the conceptual model level results in a lack of evolvability. He points out that some events may change the state of some facts in the real world. A property may become a relationship or vice versa. For example, in a hospital information system, if we had a situation where a particular 'sister' is responsible for a particular 'ward', we could then say that the attribute 'sister in charge' of a ward "X" is sister "Y". If the hospital rule changed to a situation where a 'sister' can be responsible for many wards and a 'ward' can be the responsibility of many 'sisters' then the attribute 'sister in charge' does not apply any more. Now, there is a relationship 'in charge of' between 'sister' and the entity 'ward'. The data manipulation functions which deal with these facts will not work any more. Falkenberg has introduced a simple model with two basic concepts, namely object and role. There is no distinction between relationships and properties, but the model allows semantically irreducible n-ary associations.

A rather unusual approach, was put forward by Grotenhuis (1976). His conceptual model consists of a family of relations called info-spaces, which are defined on component domains, and a family of relations between info-spaces called info-structures. The concepts of info-spaces and info-structures are similar to the concepts of entities and relationships respectively.

Interpreted Predicate Logic (IPL) by Gray (1984) uses two concepts for modelling information, namely, entities and ideas. The concepts of attributes and entity types are not recognised. A relationship in other approaches is replaced by a proposition about entities. An entity set is replaced by a proposition about two entities. IPL defines the conceptual model in a formal language based on predicate logic. This is a set of expressions which represent the logical meaning of the data and the constraints that are placed on this data.

Orman (1984) has based his Familial Model of Data on the algebraic theory of sets. Each data model is a collection of sets and families of sets, from which the name derives. Each set is a named collection of objects playing a unique role specified by the name, as for example, student or course. A family of sets is created by indexing one set by another and partitioning one of the sets. Thus each set of the family contains the students enrolled in a particular course. The model can describe object membership and relationships. Two types of abstract hierarchy are allowed - generalisation, involving subset and superset relationships and aggregation where several relationships are treated as one.

Most people treat the conceptual model as a finite, varying collection of entities and associations. The importance of having a conceptual model that evolves through time has been discussed. Nothing has been said about how such a model itself should treat the concept of 'time'. This can be included by introducing a time domain in the model. Some models keep different time versions of the same object system and deal with it as a change in relationships or properties. Benci et al (1976) incorporate time in the conceptual model by introducing data items or time domains in relations where necessary. Bubenko (1977) deals with the temporal dimension by introducing new classes of entities known as information objects which serve as a means of storing knowledge about the current states of associations. He feels it is not practically feasible to hold all the time versions of an association in the conceptual model. Information about an association should be represented in the model by a finite number of information objects. In systems where it is necessary to store historical information, this approach is not enough. Bubenko suggests that one way to overcome this problem is to represent, by stored information objects, all statements about external events which affect the model. Time will be a parameter for these statements.

Sundgren (1978) points out the importance of the dynamic aspects of the model for advanced control systems. The basic building element of his model is the e-message which gives details about an elementary situation in the system. The e-message has three components <o,a,t>, where 'o' is a reference to a set of objects (entities), 'a' is a reference to a relationship type or a property value, and 't' is a reference to a time when this relationship or property is valid.

After reviewing a number of conceptual models it is difficult to conclude which is the best model. McGee (1976) points out that the implementation criterion of the model is very important to the user. The model should not be difficult to implement but on the other hand should give a good performance. Not all the criteria for a good conceptual model may be met at the same time. One criterion can be met only at the expense of another. If a user wants a simple implementation, he might have to forgo some other objective to achieve it. It is up to the to the data base administrator to decide what is important to his needs and choose the evaluation criterion of the model.

As for approaches considering entities and relationships, we have Entity-Relationship modelling, binary relationships, interpreted predicate logic and the familial model approach. The latter two are difficult to model but are more powerful and support more semantic information. Interpreted predicate logic can be difficult to translate into a DBMS because it is difficult to map this to a record structure, which is the structure supported by most DBMS. The familial model is difficult to use as it relies on complex mathematical theory. However it produces natural and semantically rich models, supports diverse user views and includes a direct route to a DBMS via its own data retrieval language.

The Entity-Relationship approach is popular because of its direct mapping of attributes, entities and entity types on to a DBMS's fields, records and record types respectively, and is easy to model. However, it does not model dynamic business aspects and has an inflexible approach to entities, attributes and relationships. In both entity and binary relationship approaches, the types of rules supported are restricted.

1.3.2 Mapping

Once a conceptual model that supports all users' views is obtained, it is necessary to map this data representation on to a data structure of the target DBMS. The mapping should be such that there is no loss of information and the original processing specifications are satisfied. This phase is dependent on the logical structure of the target system and is termed the logical mapping. Conceptual modelling and logical mapping are based on Information Structure Perspective (ISP). When the target DBMS has been selected it is necessary to analyse the physical properties of the DBMS and the physical mapping is governed by the physical data modelling features available in that system and the usage pattern of the data. This is based on Usage Perspective (UP) data.

In general it can be said that we should be able to map the conceptual model on to any physical structure dictated by the target system. This involves mapping the structured conceptual model on to storage structures of a particular DBMS and defining access methods, depending on the options available in that DBMS. The objective is to optimise the data access for the applications so that the functions are accomplished in an acceptable period of time. This process involves dealing with problems like file structuring and access path selection. The performance of a system is affected by the accuracy of the mapping.

Lusk and Overbeek (1981), Sakai (1980) and Tsao (1980) have outlined a few steps involved in translating Entity-Relationship diagrams into hierarchical and IMS logical DBMS. Current research is oriented towards automating the data base design process or designing tools which will aid data base design, particularly using relational DBMS.

1.4 Design aids

The process of data base design can be very time consuming, particularly for large and complex systems. A survey performed by Crocker (1984) shows that analysts find the technical skills required for file design, file organisation and file access the most important to system development. Whether we accept the result of this finding or not, it is evident that file design or data base design is a crucial stage in systems development and the time spent by analysts at this stage needs to be considerable.

Tools are being developed to aid data base design. Howden (1982) in his survey, finds that tools usually offer graphical aids to document conceptual models and some automated functions which reduce the development time and enforced standards. Martin and McClure (1985, pp 2) point out that some of the tools have made the development process more productive. They argue that

> "The computer enforces discipline and permits types of cross-checking, calculation, and validity checks that human beings often do not apply".

Brown (1982) finds that most of these tools are being used by the designers.

Reviewing the arguments put forward by Howden (1982), Brown (1982), Martin and McClure (1985), Parkin et al (1987) and many others, it can be concluded that tools to convert the analyst's data model automatically into an information system on a particular DBMS will have several effects, viz:

a) Reduce development time and cost for information systems.

b) Minimise the possibility of the analyst's model being misinterpreted because he will create the system rather than the programmer.

c) Offer the analyst various software tools for structuring the data model.

d) Ensure that a specific method is being followed and that definite steps are being taken in a design process by providing guidelines in what can be a very complex process.

e) Introduce and enforce standards if they are used consistently.

f) Improve communications between analysts, designers, users by providing a common, unambiguous means of communication.

Argus, described by Stucki and Walker (1981), is a system that supports the creation and modification of data flow diagrams and produces structure charts for design.

Braegger et al (1985) have developed an interactive data base design tool called Gambit. It is based upon an extended relational Entity-Relationship model. The description of data structures and maintenance of consistency are aided by using Modula/R (a data base programming language). An interesting aspect of Gambit, in terms of data analysis, is that it allows design testing of data bases via a prototype facility. The prototype is a model of the intended data base and provides the facility of checking that the designed data base will fulfil the stated requirements.

Chen (1980) has developed a graphic based system that aids the generation of E-R diagrams. The system provides operations for creating and manipulating objects in the model. The system offers two types of operations - one type to deal with the creation and manipulation of objects in the model and the other for controlling the display of the model. The purpose of the system is to reduce time spent on drawing and redrawing models. The tool is intended for designers familiar with E-R modelling.

GDOC by Ferrara and Batini (1984) is a tool for computer-aided design of data base applications. It provides an interactive specification of conceptual models and automatic generation of physical data bases for dBase II environments. Graphics facilities are available for drawing E-R models and the automatic generation of physical data bases provides useful prototyping facilities.

Information Systems Designer (INSYDE) by King and McLeod (1985) is an experimental software system which guides the designer through the conceptual model design, but it is not very useful as a general data base design tool as it is oriented towards office systems applications.

Data Base Design described by NCC (1980) is a computerised tool which may be used for view integration and DBMS mapping. It is an IBM product aimed at its hierarchical DBMS (IMS). It can also be used to analyse data requirements for completeness, redundancy and consistency checking.

August, by Davis and Arora (1985), is an experimental expert system which can be used to translate a conventional file system into a conceptual E-R model which can then be translated into a commercial data base. It is a set of modules written in Prologue and at the time of writing worked only on Cobol files.

Williams et al (1986) have developed a system to assist a data base administrator in the tasks associated with the restructuring of simple data bases. First, the pattern of usage is monitored by logging every query. This information is used to produce a set of statistics. The statistics are then used to detect areas where data base restructuring would improve performance.

Currently the trend of research is toward developing Integrated Project Support Environments (IPSEs), which are intelligent, knowledge based tools which will advise designers about the method to be used, help with design decisions and automatically perform the tasks within the design phase. Majic, by Sutcliffe et al

(1987) and Intellipse, by Bader et al(1987) fall into this category.

Whilst the ultimate objective of any work involving IPSEs is to automate all the stages of systems design, current efforts are limited to automating various stages of the cycle. Although these tools are contributing towards the design of a system, the full potential of the tools will only be realised when they are integrated with other tools which automate the complete life cycle of a system design project. The process of integration will be successful if these tools adhere to similar standards. Esprit's Portable Common Tool Environment (PCTE) (1986) is aiming to set these standards. Many IPSE manufacturers are developing their systems to adhere to PCTE's interface.

Tools to support particular design methodologies are being developed by various commercial organisations. IEF from James Martin Associates, IEW from Arthur Young Inc, AUTOMATE PLUS from LBMS, are examples of such tools.

1.5 Objectives of this research

Wasserman (1980) describes a data design methodology as a collection of tools and techniques which can be applied normally to successive data base development projects. Boehm (1976) and Wasserman (1980) point out that many of the data base design methodologies known are based on software design methodologies.

The overall aim of this research work was to develop computerised tools to aid a data base design methodology. As there was still no agreement on which was the best approach for data base design, it was decided to choose an approach which was based on accepted principles. The design approaches of Kahn (1976, 1978), Tozer (1976), Chen and Yao (1977), Gerritsen (1975,1978) and CACI (1980), and the view modelling and integration concept of Navathe and Schkolnick (1978), Navathe and Gadgil (1980) and data analysis methods of Davenport (1978), Palmer (1978) and CACI (1980) have influenced this research.

The tools were not intended for data base users, but professionals who will be involved in designing and implementing a data base. The users of the tools will include analysts, data base designers, data base administrators, etc. In this report, the terms analyst, designer and data base administrator have been used to refer to these users. It was intended that the tools would be able to do the following:

1. Reduce the time consumed in repetitive, laborious tasks such as modifying models, updating documentation, etc., thus saving development cost and time.

2. Cross check different parts of the design and check for consistency, thus reducing the possibility of human errors.

3. Simplify the design process so that the tools can be used by non-experienced designers for data base design.

To be able to fulfil the specified goal, four primary objectives were set. The first was to study the various data modelling techniques proposed and then to select one which best suited the objectives which follow. A slight variation of Chen's Entity Relationship (E-R) approach was chosen because it captured the structural constraints supported by the types of popular DBMS, namely relational and CODASYL.

The second objective was to examine the factors that governed the mapping of the conceptual model on to any logical model. This phase is a formal process and thus it is possible to formulate rules for mapping, if the logical structure of the target data base system is known. It was hoped that it would be possible to define formal rules that would be involved in mapping an E-R conceptual model into the data structures of the well known DBMS.

The performance of the implemented system depends very much on how well it is physically designed. The performance can be modified by influencing the physical structure of the data base to suit the users' needs. Gane and Sarson (1979), Orr (1976), Yourdon and Constantine (1979) advocate that minimisation of I/O time is an important issue to be considered in this phase.

The third objective of this research was to be able to specify an effective solution for organising the data base files depending on usage. A similar concept has been applied by Hoffer (1975), Hoffer and Severance (1975) and Clarke and Hoffer (1979) in their record partitioning problem. The steps in the design process specified so far are free from any particular implementation. The next phase of the data base design is to implement physically the resultant data structure obtained from the previous phases on to a particular DBMS package.

The fourth objective was to study the logical and physical characteristics of a relational type and a CODASYL type DBMS and define algorithms to map the data model *obtained on these* target DBMS. It was hoped that it would be possible to evaluate the performance of the implemented data base system.

1.6 How far the objectives have been met

Various data modelling techniques have been examined and a variation of Chen's E-R approach (see section 2.4) was chosen to represent our data model. This technique was used at the George Eliot Hospital, Nuneaton, to determine the information structure of part of a hospital. This data model was used to test the logical and physical mapping carried out in the following phases. Interactive programs were written which would communicate with the data analyst and store the result of the data analysis in the form

NO PAGE

27

base can be optimised. Much will depend on the nature of the project or application. Application development time, execution time, data storage, user level of expectation and data flexibility have been identified by Inmon (1981) as some of the options on which a data base design can be optimised. He also points out that optimisation of data base design on storage and users' level of expectation involves creating complex data structures which in turn introduces complexity to the application programs. Over-optimisation of development time usually results in problems during implementation. In this work emphasis was put on producing a flexible system which was efficient in supporting the known functions. The performance of the system can be influenced by organising the data to suit the usage pattern. Organising the data to allow fast access for the most critical processes so as to improve the performance of the data base was the chosen option for this research. The details noted about the functions were used to make decisions on how to organise the files and what access methods to use. Programs were written which took these factors into consideration and assigned the best feasible support to the paths which were most frequently used.

The physical and logical characteristics of relational DBMS, such as INGRES and MIMER, and CODASYL types, such as VAX-11 DBMS, were studied. Rules for mapping the conceptual models on to these DBMS were formulated. The resultant relational data model obtained from the previous phases were implemented on INGRES and MIMER. Programs were written which took the logical model and the physical characteristics of the access paths to produce the data definition file description for each of the relational DBMS. Algorithms have been defined for mapping to a CODASYL DBMS.

1.7 Outline of the structure of the thesis

Chapter 2 describes the data analysis technique used in this project. It also describes how this technique was applied in a hospital environment to form the conceptual model for part of the hospital and contains the description of the programs (ENTEST.P and RELTEST.P) that record the results of the data analysis.

Chapter 3 describes a functional analysis technique and how this approach was applied in the hospital environment so as to deduce the usage pattern of the data. It also describes the programs (FUNCTEST.P and ANALYSIS.P) that record the result of the functional analysis and examines this result to produce the usage pattern of the entities, attributes and relationships.

Chapter 4 covers logical design which considers the side effects of updating an E-R model. This chapter discusses the factors that govern the mapping from the E-R model to any target DBMS logical model.

Chapters 5 and 6 discuss the issues for mapping an E-R model on to a relational and a CODASYL type data model respectively. They describe the algorithms for performing these operations.

Furthermore, they describe the programs (RELMAP.P and CODMAP.P) that follow these algorithms and produce relational and CODASYL models.

Chapter 7 discusses the factors that govern the physical storage structure of data. Different storage options are analysed and an automatic labelling algorithm is deduced which produces an efficient storage structure for the data, depending on usage pattern. The description of the program (LABEL.P) which performs the labelling is also included.

Chapter 8 describes two relational DBMS, namely INGRES and MIMER. The factors that govern the implementation of the relational model on to these DBMS are considered and an algorithm is put forward which facilitates these implementations. Programs (ING.P and MIM.P) perform the implementation.

Chapter 9 describes a CODASYL type of DBMS, namely VAX-11 DBMS and an algorithm is put forward which can be used as a software tool to facilitate the use of CODASYL DBMS.

Chapter 10 is the concluding chapter which discusses the contributions of this study and areas for future work.

CHAPTER 2

DATA ANALYSIS

2.1 Introduction

A computer based information system should provide users with relevant information in a form that is comprehensive and accurate. In order to achieve this, it is necessary to perform a systematic analysis of the structure of the organisation and the information requirements of the organisation for which the system is being designed. Such an analysis yields the conceptual data model.

Kent (1978) argues that data models are ineffective in representing information about the real world. He argues that data models are rather inflexible and fail to portray the subtle differences in the complete meaning of the real world. The author of this report recognises that though one would like a model that captures the complete meaning of the real world, such a model is not feasible. The model has been used to capture appropriate amount of meaning which is relevant and appropriate for the purpose required. For this research, the data model is used to structure data so that it can be manipulated by a computerised information system.

2.2 Conceptual data model

A conceptual data model describes the data resource of the organisation and also specifies how the data should be used. The data model should be independent of the computer system which is used to store and access the data. The model should not only include the current view of the data but also how it might be used in the future.

Various conceptual data modelling techniques exist and they were discussed in the previous chapter. A variation of the E-R approach of Chen (1976) was chosen to represent our data model. Chen's model is widely used in the commercial world. It is independent of any particular implementation, but captures the structural constraints which could be supported by the relational and CODASYL types of DBMS.

The concept of a weak entity, that is, an entity which may depend on other entities for it existence, and entities with ID dependency, that is, entities which cannot be identified by their own attributes, have not been used for this project. Instead, the concept of membership class has been introduced. The explanation for membership class is given in section 2.3.2. The concept of membership class was necessary to reduce the number of relations obtained in the relational model and to decide the insertion/retention clauses for the CODASYL model. Moreover, it dealt with the concept of weak entity and ID dependency.

2.3 Chen's Entity-Relationship(E-R) model

The basic components of Chen's E-R model are the entity and the relationship. An entity can be a person, concept, event or any object that can be distinctly identified. A relationship defines the association between entities. Entities with common properties are classified into entity sets. The role of an entity in a relationship is the function performed by the entity. For example, 'FATHER-SON' is a relationship between two 'PERSON' entities, 'FATHER' and 'SON' are roles of these entities in the relationship. Information about properties of entities and relationships are expressed in terms of attributes and values. Values are classified into different value sets, such as 'COLOUR', 'INCHES' etc. An attribute is the name given to the mapping between an entity and a value. For example, an attribute of entity set 'PERSON' may be 'COLOUR OF HAIR' which maps to the value set 'COLOUR'. An attribute or a group of attributes which provides a one-to-one mapping from the entity set to the corresponding group of value sets is known as the entity key. Entities which cannot be uniquely identified by their own attributes but use their relationship with other identifiable entities as a form of identification are known as weak entities. For example, a 'TOWN' may not be uniquely identified unless the 'COUNTY' is stated. Furthermore, entities which depend on other entities for existence are said to have existence dependency on others. Most weak entities are associated with existence dependency but existence dependencies are not necessarily associated with weak entities.

2.4 Modified Entity-Relationship(E-R) model

In our Entity-Relationship model, real world information is represented by entities and relationships. Objects of interest are known as entities which can be classified into entity sets or types having the same properties. Entity types are associated with each other by relationship types. In this report entity types and relationship types are often referred to as entities and relationships respectively. Any number of entity types may participate in a relationship type. Relationships between two entity types are adequate for modelling most real situations. Moreover, all relationship types between more than two entity types can be broken down into a number of binary relationships, and the original relationship type can be represented by an entity type. For example, the relationship between 'SUPPLIER', 'PART' and 'CUSTOMER' (Figure 2.1) can be represented by an entity type 'CONTRACT' and three binary relationships between entities 'CONTRACT' and 'SUPPLIER', between entities 'CONTRACT' and 'PART' and between entities 'CONTRACT' and 'CUSTOMER' (Figure 2.2). In this project, most of the relationship types have been represented as binary relationships. However, the mapping rules designed are able to handle relationship types which are not binary in nature.

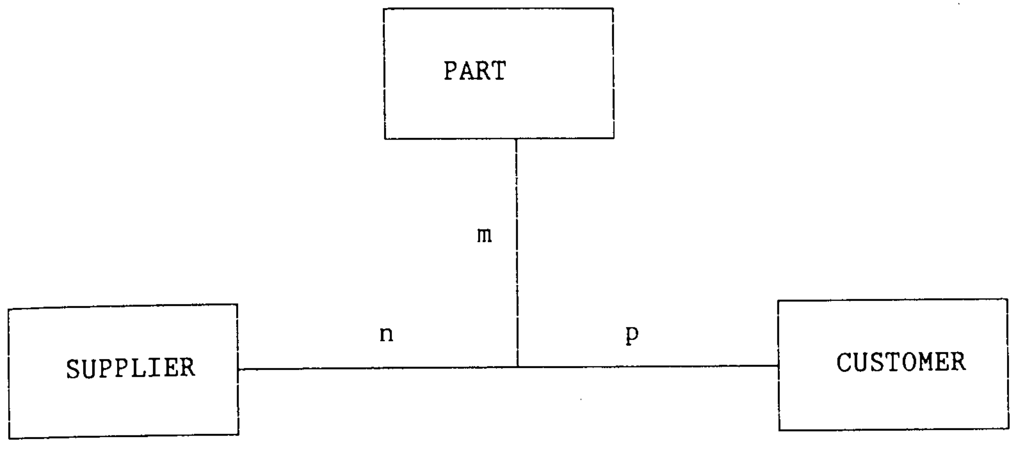


Figure 2.1 N-ary Relationship

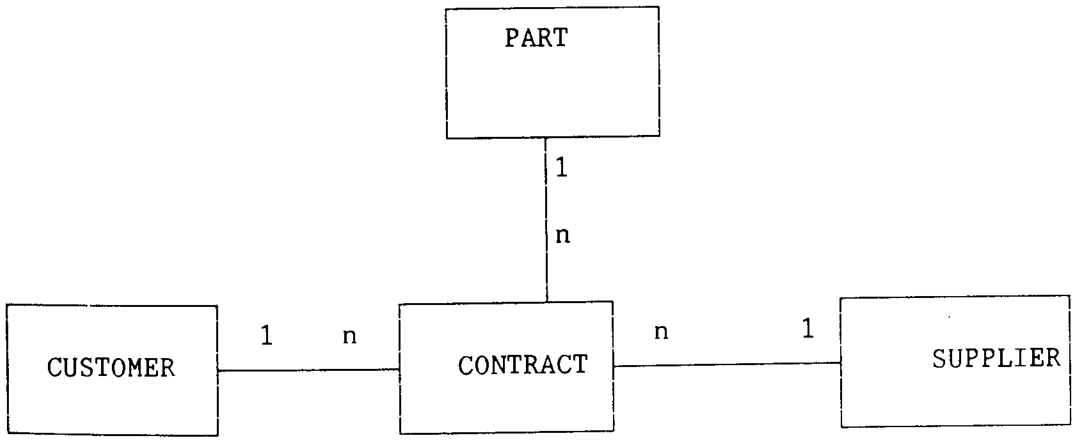


Figure 2.2 Binary Relationships

A uniquely identified entity from an entity type is known as an entity occurrence. For example, DR. COX is an entity occurrence of entity type 'CONSULTANT'. An instance of a relationship type is specified when the individual entity occurrences taking part in a relationship type are known. M.SMITH is 'TREATED BY' DR. COX is an instance of the relationship 'TREATED BY'. Attributes define the properties of entities as well as relationships. 'Speciality' is an attribute of the entity type 'CONSULTANT' describing his field of expertise. The attribute or group of attributes that uniquely identifies an entity occurrence is called the identifier of the entity type. 'Emp-no' is the identifier for the entity type 'CONSULTANT'. The identifier of a relationship is formed from the identifiers of the entities participating in the relationship, hence the identifier of the relationship 'TREATED BY' is 'emp-no,pat-no'.

2.4.1 Degree of a relationship

There are three types of degree of a binary relationship. They are one-to-one (1:1), one-to-many (1:n) and many-to-many (m:n). If Rxy represents a relationship between the two entity types Ex and Ey, then we can specify the degree of Rxy in the following ways.

One-to-one:- Rxy will be of degree 1:1 if an occurrence of Ex is associated with one occurrence of Ey at the most, and an occurrence of Ey is associated with one occurrence of Ex at the most.

One-to-many:- Rxy will be of degree 1:n if an occurrence of Ex is associated with more than one occurrence of Ey and an occurrence of Ey is associated with not more than one occurrence of Ex.

Many-to-many:- Rxy will be of degree m:n if an occurrence of Ex is associated with more than one occurrence of Ey and vice versa.

The degree of a relationship type which shows association between more than two entity types Em, En,.......Ez can either be represented as 1:1:.......:1 or m:n:......:z. The first option implies that one occurrence of Em is associated with one occurrence of each of En,.......Ez. The second option implies that one occurrence of each of the entity types are associated with more than one entity occurrence of the other entity types. In practice, it seems that most non-binary relationships have a degree of m:n:......:z.

The features of E-R models described so far are the same as those described by Chen (1976). Date (1986) argues that Chen's E-R model is not a data model but "nothing more than a collection of data structures (actually a collection of relations of various kinds); in other words, the manipulative and integrity aspects are virtually ignored". He states that the popularity of an E-R model is due to the diagramming technique used to represent the data structures. The concept of 'membership class' was introduced in our model to address the manipulative and integrity aspects.

2.4.2 Membership class

Howe (1983) describes two ways in which an entity can participate in a relationship. From the analysis of the organisation, we can sometimes specify that every occurrence of an entity type must participate in a certain relationship type. An entity occurrence of that type cannot exist if it does not take part in that relationship. The membership class of that entity type is classified as obligatory for that relationship type. On the other hand, if it is not necessary for every entity occurrence of an entity type to participate in that particular relationship type, then the membership class of that entity type is classified as non-obligatory for that relationship type. The obligatory membership can explain the concept of 'weak entity'. An entity Ex will be a weak entity if its existence depends on another entity, say Ey. To represent this information in this model we can set up a relationship Rxy between the entities Ex and Ey, and state that the membership class of the entity Ex is obligatory in that relationship. Entity type Ex will not be able to exist unless it takes part in the relationship Rxy. An example is given in section 2.3.

Entities which cannot be uniquely identified by their own attributes but use their relationships with other entities as their form of identification, are said to have 'ID dependencies'. For example, entity 'CLINICAL TIME SCHEDULE' needs the identifier of 'CLINICAL SESSION' to be properly identified. The attribute 'TIME' on its own would not identify an occurrence of CLINICAL TIME SCHEDULE'. It is felt that there is no reason for introducing a special type of entity for ID dependency. Instead, the identifiers of those entities can be formed by extending the identifiers of the entities that they are related to. In our hospital environment, 'CLINICAL TIME SCHEDULE' is such an entity and its identifier is CLINIC-NO, DATE, TIME. Its dependence on 'CLINICAL SESSION' is represented by the fact that its identifier is formed from the identifier of 'CLINICAL SESSION', that is CLINIC-NO, DATE. The membership class of the entity with the borrowed identifier always has obligatory membership in this relationship.

A similar concept to membership class has been described by Robinson (1985). He used the terms mandatory and optional. These terms are similar to those used in CODASYL models. To avoid confusion, it was decided to use Howe's terminology. The use of the membership class will be obvious when the rules for mapping the E-R model on to relational and CODASYL structures are defined.

2.4.3 The diagrammatic model

In our hospital data base 'PATIENT', 'CONSULTANT', 'CLINICAL SESSION' etc., are examples of entity types. 'PATIENT TREATED BY CONSULTANT' and 'CONSULTANT IN CHARGE OF CLINICAL SESSION' represent the relationships between the entity types 'PATIENT' and 'CONSULTANT' and 'CONSULTANT' and 'CLINICAL SESSION' respectively. If the organisation rules are such that a patient could be treated by more than one consultant and a consultant can treat more than one patient, then the degree of the relationship 'TREATED BY' is m:n. For the other relationship example, if we found that a consultant can be responsible for more than one clinical session, but a clinical session can only be in the charge of one consultant, then the degree of the relationship 'IN CHARGE OF' is 1:n. Moreover, if the organisation rules state that any patient should be treated by one consultant at least, but it is not obligatory for a consultant to treat a patient, then the membership of the 'PATIENT' entity type is obligatory and the 'CONSULTANT' entity type is non-obligatory in the relationship 'TREATED BY'. All this information can be represented as an Entity-Relationship (E-R) diagram (Figure 2.3).

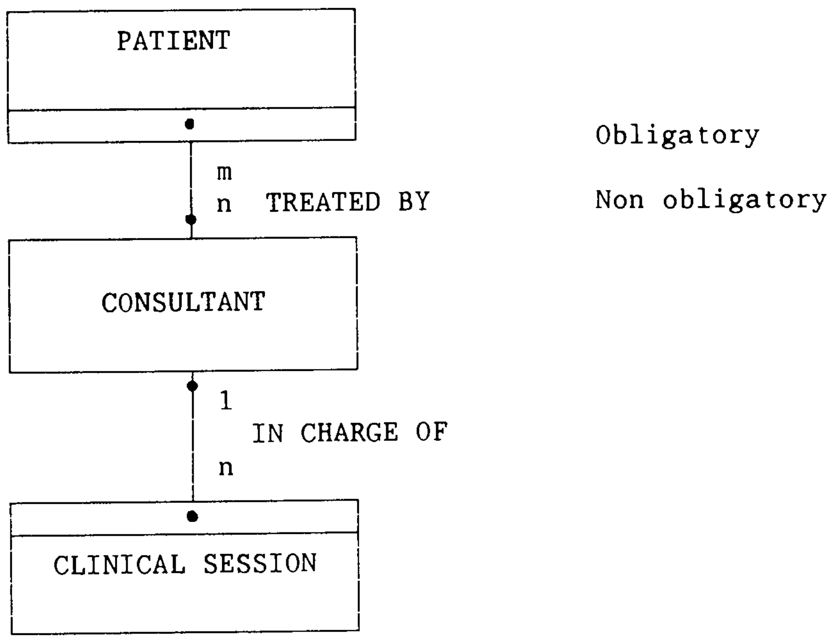


Figure 2.3 An E-R Diagram

2.5 Data analysis

Data analysis consists of a series of techniques for identifying and analysing the data elements of an organisation and arranging them to form a structured data model (conceptual model). To derive a conceptual model, which is represented as an E-R model, we would need to identify the entity types, the relationships between these entities and the attributes of the entities and the relationships.

The phases to be followed for data analysis in this project were adopted from CACI (1980) and Rock-Evans (1981) and are as follows:

Preliminary.

Define the area for analysis.

Entity modelling.

Identify the entities.

Identify the relationships between entities.

Consider entity identifiers.

Detailed analysis.

Identify attributes of each entity.

Check for hidden or redundant relationships.

Refine the entity model.

The data analysis process described above was applied in a hospital environment to formulate the conceptual model for part of a hospital. The hospital is used as an application area to illustrate the use of the tools constructed in this research.

2.5.1 The application area

A hospital information system is a communication and data processing system which receives, stores and processes data relevant to patient care (clinically related functions), patient administration and hospital management. The objectives of a hospital information system are to:

- improve quantity, quality, utility and speed of medical data communication
- communicate individual patient data amongst the professionals providing medical care and hospital service and administrative departments
- perform scheduling and booking of various hospital resources
- store data for administrative and business functions
- support clinical and health services research
- provide planning for hospital and medical services.

Better information is the key component for achieving the objectives identified above. Many application-oriented data files are not sufficient for the information flow. Data bases that serve many of the application areas in the organisation are more appropriate. The problem of communicating information is not unique to a hospital information system. The objectives of information systems in other disciplines are very similar, the key being the timely delivery of relevant information to the relevant people.

2.5.2 Data analysis for the hospital

The data analysis approach discussed in the previous section was carried out at the George Eliot Hospital, Nuneaton. This is a hospital in a small industrial town. It supports everyday patient care, but patients requiring sophisticated treatment and diagnosis are sent to the nearest teaching hospital. The hospital is partitioned into departments according to the job they fulfil. The departments fall into two categories:

1) Patient care and patient administration, and

2) Organisational management.

Patient care includes departments such as outpatients, medical and surgical wards and pathology laboratories. Personnel, housing and the hospital store are included in the organisational management category.

In the first stage of the analysis, the purposes of the information system were established. It was decided to look into the area of patient care and administration. Information concerning organisational management could be incorporated into the system at a later stage.

Having set the scene, it was necessary to decide on the areas to be examined. These will be referred to as 'data areas'. It was important that these areas were discrete, had definable boundaries, were independent of specific applications and were small enough to be manageable. The different departments were the obvious choice for the data areas and proved to be satisfactory.

Once the areas to be analysed were defined, the next step was to determine the crucial entities and the relationships between them. The entities and the relationships were identified by interviewing the head of the department and also by studying the existing documents which consisted of numerous hand-written forms. Initially, the obvious entities and relationships were recognised, but they did not produce a complete entity model. Data analysis is an iterative process and it needed a few iterations before a complete picture was obtained. Only when a satisfactory model was obtained were the attributes considered. The model was checked for redundant and hidden relationships and refined until a satisfactory final version was achieved.

Care was taken that all known elements were defined as entities or attributes. An object is classified as an entity if there is some information about it, it is of significance itself, it has a means of identification and there are connections between it and other entities. If there is any doubt about any object, it is safe to classify it as an entity and, after further analysis, it will become apparent whether or not the right choice was made. The analysis for one data area (the Record Office) is described in detail. For the other data areas, the entity models obtained are described in Appendix A.

2.5.2.1 The Record Office

The Record Office was chosen as the first data area to be analysed. The Record Office was chosen as the data area which was analysed first because it was apparent that this department was concerned with information which overlapped with the other areas under analysis. The entities obtained for this area gave some indications of the entities to look for in the other data areas. This department is concerned with the basic patient record, i.e. patient identification, together with clinical information such as diagnosis, operations performed, nursing reports and medical history. The department is also responsible for maintaining the lists of patients awaiting admission to the hospital or the outpatients' clinics. The main objective is to enable consultants to select patients from their waiting lists in a way that efficiently utilises the hospital resources and minimises delay. The selection technique depends on the priorities assigned to patients, depending on the seriousness and urgency of the patient's condition.

The entity types obtained from this department are:

PATIENT

CONSULTANT

DISEASE

OUTPATIENT WAITING LIST

CLINICAL SESSION

CLINICAL TIME SCHEDULE

BOOKINGS FOR WARDS' BED

SURGICAL SESSION

SURGICAL WAITING LIST

ADMISSION WAITING LIST.

The relationships between these entities are,

CONSULTANT AND PATIENT	m:n
CONSULTANT AND CLINICAL SESSION	1:n
CONSULTANT AND SURGICAL SESSION	1:n
PATIENT AND DISEASE	m:n
PATIENT AND SURGICAL SESSION	m:n
PATIENT AND CLINICAL SESSION	m:n
PATIENT AND CLINICAL TIME SCHEDULE	1:n
CLINICAL SESSION AND CLINIC TIME SCHEDULE	1:n
PATIENT AND OUTPATIENT WAITING LIST	m:n
PATIENT AND SURGICAL WAITING LIST	m:n
PATIENT AND ADMISSION WAITING LIST	m:n
PATIENT AND BOOKINGS FOR WARDS' BED	1:n
CONSULTANT AND OUTPATIENT WAITING LIST	1:n
CONSULTANT AND SURGICAL WAITING LIST	1:n
CONSULTANT AND ADMISSION WAITING LIST	1:n
CONSULTANT AND CLINICAL SESSION	1:n
CONSULTANT AND SURGICAL SESSION	1:n

The attributes for the entities identified are as follows:

PATIENT(pat-no, pat-name, pat-address, pat-category, sex, date-of-birth, marital-status, next-of-kin, blood-group, X-ray-information, reference, allergy)

CONSULTANT(emp-no, name, address, speciality)

OUTPATIENT WAITING LIST(W/L-no, speciality, selection-criteria)

CLINICAL SESSION(clinic-no, date, start-time, finish-time, est:-new-pat, est:-repeat-pat, new-pat-booked, repeat-pat:-booked, speciality)

CLINICAL TIME SCHEDULE(clinic-no, date, time, free/booked, new/repeat)

BOOKINGS FOR WARDS BEDS(bed-no, ward-code, date)

SURGICAL SESSION(session-no, date, start-time, max:-major-cases, max:-minor-cases, major-cases-booked, minor-cases-booked)

SURGICAL WAITING LIST(S-W/L-no, speciality, selection-criteria)

ADMISSION WAITING LISTS(A-W/L-no, speciality, selection-criteria)

The membership class of the entities taking part in the relationships were also considered and is described in the E-R diagram (Figure 2.4) drawn from these entities and relationships.

The other data areas analysed were the Pathology laboratory, Pharmacy department, X-ray department, Wards and the Nursing department. The entity models for these data areas are described in Appendix A.

2.5.2.2 Merging data areas

Once the data areas were analysed, it was necessary to integrate them to form a global entity model (Appendix B) which satisfied all data areas. During this phase the inconsistencies in the data areas were removed. These inconsistencies may have risen either from one name referring to different components (homonyms). Thus, for example, 'STAFF' can mean consultants, nurses and administrative people. Inconsistencies could also have risen where different names refer to the same components (synonyms). The Pharmacy was frequently referred to as the Pharmaceutical department. Entity models were integrated by superimposing the identical entity types on the different entity models. The entity types 'PATIENT' and 'CONSULTANT' appeared in most of the data areas, so it was possible to merge the data areas by superimposing the data models on these entity types. The global entity model does not show the complete list of entities, but only those entities which are used to superimpose the different entity models. The bold outlines in Figure B.1 show the various data areas.

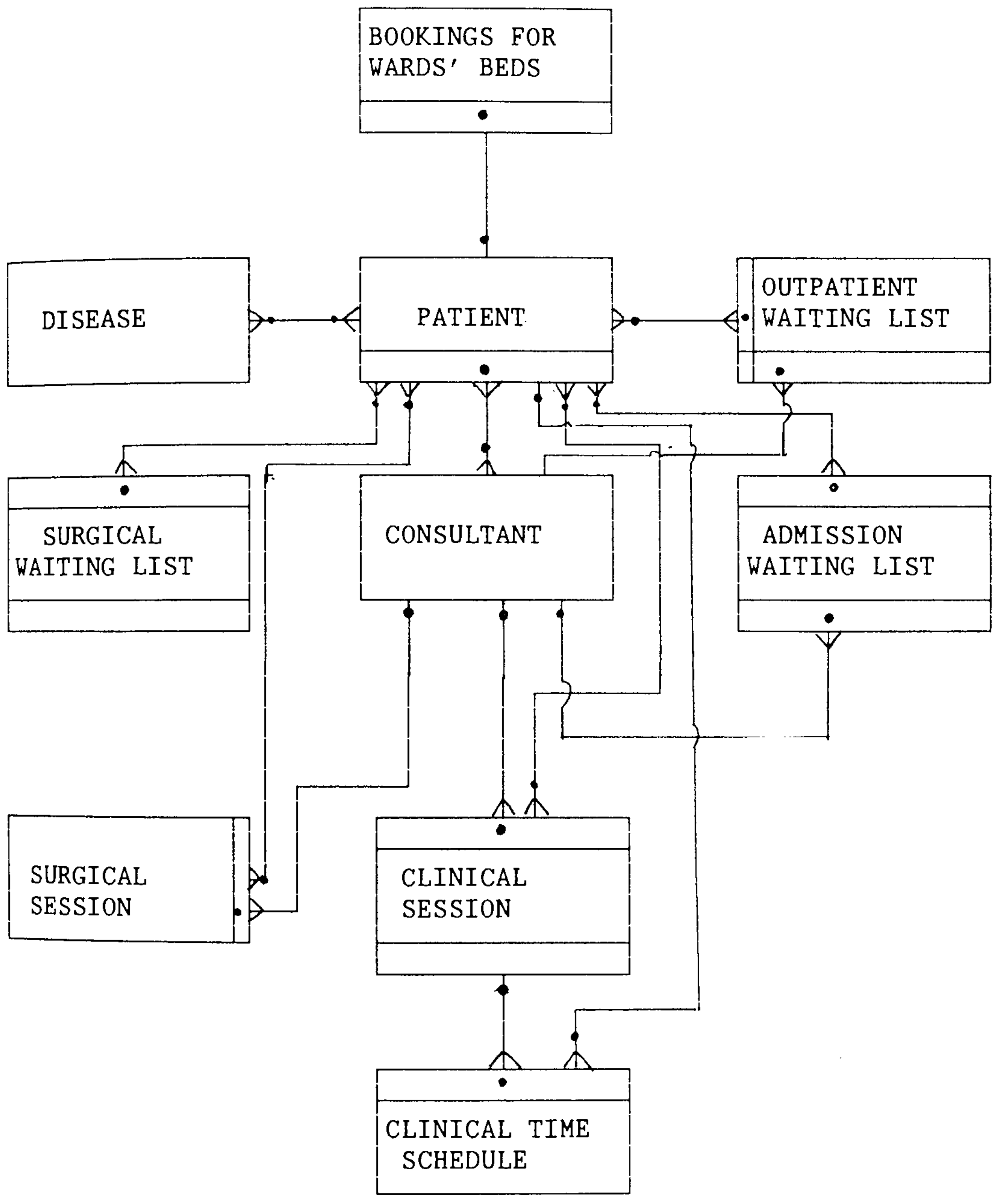


Figure 2.4 E-R Diagram for the Record Office

2.6 Storing the result of the data analysis

All the entities and attributes which were identified in the analysis stage were stored in a file. An interactive program, 'ENTEST.P', had been written which interacts with the data analyst and collects details about the entity. The details include factors like the name of the entity, the number of attributes each entity possesses, the primary key and the description of each of the attributes. As new entities are fed in, the program checks if the entity already exists by comparing the entity name with the entity names in the list. If it does, the program informs the analyst that it already exists and does not attach it to the list of stored entities. Otherwise, the program adds all the new entities to the list of existing entities. In the later stage of the project, information about how these entities were associated, their usage pattern, etc., were fed into the system so as to facilitate the implementation of the model on to the target data base management software.

The list of entities and their properties are stored in a file named 'entfile'. This file also contains the number of entities in the list.

The list of entities, 'entchart', is represented by an array of records named 'entity'. Each 'entity' record contains the following four components:

ename	a string of characters representing the name of the entity
keycount	an integer representing the number of attributes which form the key for the entity
noatts	the total number of attributes for that entity
entatt	an array of strings representing the names of the attributes.

The program first reads the file 'entfile' into 'entchart' and determines the number of existing entities in the file. The program then finds out from the designer the number of new entities to be inserted. For each entity the program records the number of attributes, the names of the attributes and notes the attributes which act as the identifier for the entity.

When the information about all the new entities is recorded, the program writes the new list back into the file named 'entfile'. The number of entities is also updated.

Another program, 'RELTEST.P', had been written for this phase of data base design, which interacts with the data analyst and obtains information about the relationships identified in the data analysis phase. The program gathers details about the relationships and stores them in a file. On every occasion, when a new relationship is introduced, the program documents the name of the relationship and makes sure that no other relationship with the same name exists in the file. If it does, then the program notifies this to the analyst. Alternatively, it enquires the name of the participating entities. Before it stores the name of the participating entities it makes sure that those entities already exist in the list of entities. The program also stores information about the degree of the relationship, the membership class of the entities taking part in the relationship and any attribute of the relationship if it exists. This information is obtained by interacting with the designer.

The program, 'RELTEST.P', documents the identified relationships, their degree and their membership class, and stores them in a file named 'relfile'.

The list of stored relationships is represented by 'relchart'. It is represented as an array of records named relation. Each 'relation' record consists of the following components:

rname	a string of characters representing the name of the relationship

entitya	an integer acting as a pointer to the first entity taking part in the relationship
degenta	a character representing the degree of the first entity taking part in this particular relationship
entityb	an integer acting as a pointer to the second entity taking part in the relationship
degentb	a character representing the degree of the second entity taking part in the relationship
norelatt	the number of attributes for the relationship
relatt	an array of strings representing the names of the attributes for the relationship.

The program first reads in the entities from the 'entfile' file. It then reads in the total number of existing relations (noofrelations) and the relations from the 'relfile' file. For each relation it reads in its name, pointers relating to the entities taking part in the relationship, the degree and membership classes of the participating entities, and the names of the attributes of the relationship.

Once the existing entities are read in, it finds out from the designer the number of new relationships to be added. For each new relationship, the above mentioned information is recorded. The program checks from the entity file that the entities specified do exist. Otherwise, it reports to the designer that such entities do not exist and gives the designer the chance to rectify his error.

The complete list of relationships with their related information is then stored back in the 'relfile' file. 'Formrel' is another file in which the information regarding the relationships is stored in a tabular form. The purpose of this is to make it easy for the designer to look up the relationship table.

Examples of the dialogues for typical runs of the programs 'ENTEST.P' and 'RELTEST.P' are provided in Appendix D.

2.7 Conclusion

This method for data analysis involves understanding the fundamental nature of the organisation. It is not possible to produce rule-of-thumb techniques on how to go about finding the entities and relationships. Much depends on the analyst's ability to communicate effectively with the members of the organisation. The Entity-Relationship diagrams are easy to understand and this is a good way to communicate with the user. This is a top down approach, that is, the analyst is not burdened with unnecessary details to start with. Initially his task is to identify the main entities and relationships. Only when a good understanding of the organisation is achieved, need he add the details.

As this approach requires a good understanding of the organisation, the involvement of senior management is crucial. This is always a possible source of problems because senior management may wish to keep their involvement to a minimum. This gives rise to another problem: junior staff tend to explain the organisation in terms of the functions they fulfil. It is important and difficult to separate the entity analysis from the functional analysis.

CHAPTER 3

FUNCTIONAL ANALYSIS

3.1 Introduction

The first activity in the analysis phase was to analyse the data resources of the organisation and formulate the conceptual model. The objective of such an analysis was to capture the data necessary to satisfy the processing requirements of the organisation. Therefore, having established an entity model, it was necessary to check the model against the processes and the information requirements associated with these processes. Functional analysis is a series of techniques for understanding and documenting the functions in the application area.

3.2 Functional analysis

Functional analysis is concerned with the understanding and documentation of the basic activities of the organisation. In the context of the work presented in the thesis, functions are concerned with the processing of entities. Relationships provide the access paths between the entities. The entities and relationships involved in each of the functions are represented by functional entity models. The functional entity models describe the sequence in which entities are accessed to satisfy a given function. These models are used to verify that necessary data is captured and required access paths to this data exist.

Identified functions are analysed further to derive the frequency of the functions and volume of the data retrieved, identify the critical applications and understand how records are accessed. Consolidation of such information provides details about the usage pattern of the data which is used in the physical design phase.

3.2.1 Phases of functional analysis

From the above discussion it can be stated that functional analysis can be subdivided into the following phases:

i. Functional decomposition

- identify the functions,
- decompose them until the transaction level is reached.

ii. Access path analysis

- determine the entities and relationships involved in each function,
- consider selection criteria for entities,
- estimate function frequencies,
- summarise as a functional entity model.

iii. Preparation for design

- analyse the usage of attributes by functions,
- summarise the total usage of each entity type and relationship type.

3.2.2 Functional analysis for the hospital

Functional analysis was performed for all the data areas defined and described in the last chapter. A selection of the functions performed by the record office is described here. The record office is responsible for making all the appointments for each doctor. It is responsible for dealing with the enquiries about inpatients and outpatients. It creates the surgical list for all the surgical consultants. It is responsible for providing the patient records when required.

For example, the function of fixing an appointment for a new patient from the outpatient waiting list for a consultant will be examined. This function can be decomposed into the following sub-functions:

Select consultant.
Select his waiting list.
Select a patient from this waiting list applying the selection algorithm.
Select a clinical session for this consultant.
Check whether a vacancy for a new patient exists.
Book the patient for the time scheduled for a new patient.
Delete the patient from the waiting list.
Include the patient among patients with outpatients appointment.

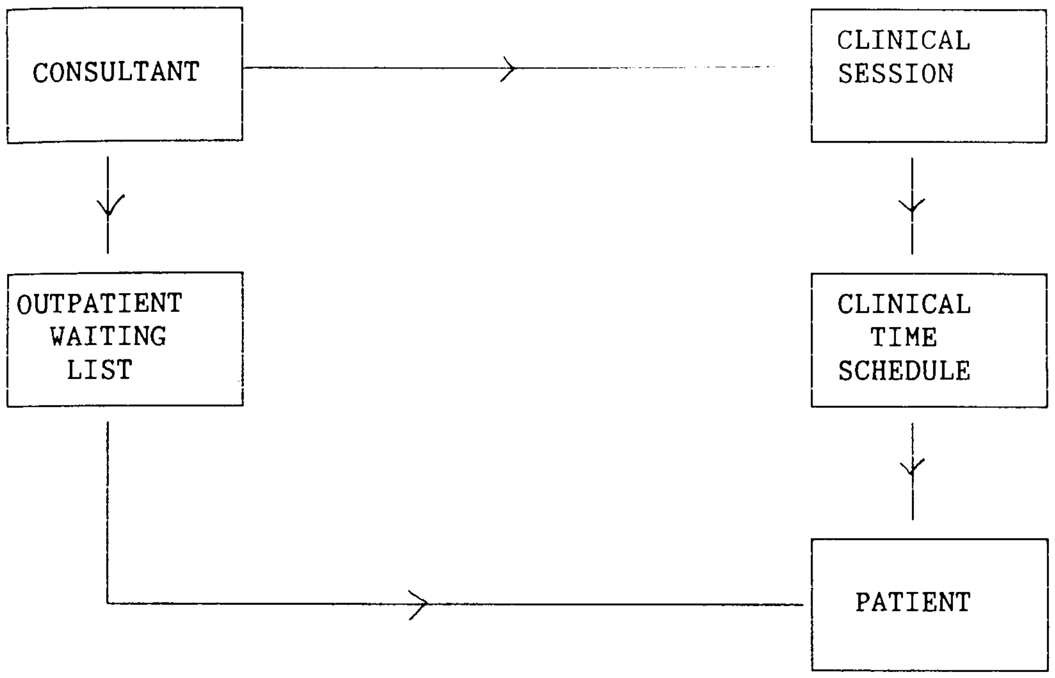


Figure 3.1 Functional Entity Model (1)

The functional entity model shows the entities and access paths which are necessary to fulfil this particular function. This model is then compared with the entity model (Figure 2.4), derived from entity analysis, to check that these entities exist and also to check that the required access paths exist as relationships between these entities. The functional entity model (Figure 3.1) shows that the conceptual model is capable of dealing with this particular user requirement.

Another function of the department is dealing with the enquiry of a patient who wants to know the time and date of his outpatient appointment. This function can be decomposed into the following sub-functions:

Check whether the patient has any appointment at all.
Check the number of the clinic the patient is booked for.
Check the time of appointment.
Check the date and other details of the clinic.
Check the name of the consultant from the clinic number.

The functional entity models (Figures 3.1 and 3.2) drawn from the functional analysis show that the conceptual model for this data area (Figure 2.4) consists of the entities and relationships required to fulfil these functions.

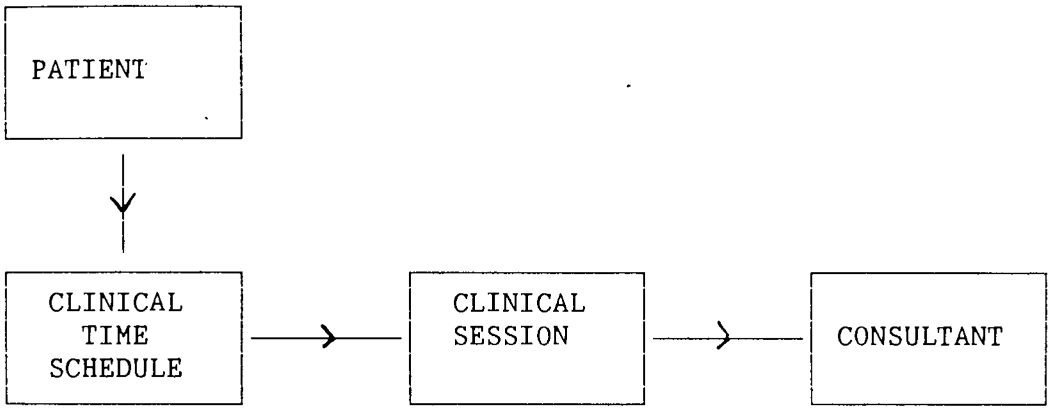


Figure 3.2 Functional Entity Model (2)

3.3 Storing the result of the functional analysis

An interactive program, 'FUNCTEST.P', communicates with the designer and records the result of the functional analysis. The list of functions, 'funcchart', is represented by an array of records named 'func'. The record, 'func', has six components and they are:

funcname	is a string of characters denoting the name of the function

funcfreq	is an integer representing the frequency of the functions in terms of number of frequencies per day
funcstatus	is an integer which can take only two values describing whether the function is primary or secondary
noacc	is an integer and denoting the number of entities accessed
entarr	is an array of record accent

The record type, 'accent', describes the entities accessed in terms of its name and the path through which the entity was accessed. The following components describe the record type 'accent'.

entname	an integer pointing to the appropriate entity in the entity list
eselectcrit	details of how the entity is selected

There are three selection options. They are 'sbyr', 'sbyp' and 'sbya' meaning select by relationship, select by primary key and select by attribute respectively. When selected by attribute, the selection clause needs to be defined. This can either be a range clause or an equality clause.

The file, 'funcfile', holds the list of functions identified by the designer. The variable, 'nooffunc', denotes the number of existing functions. When this program is executed, the file 'hospfunc' is read into an array named 'funcchart'. The program then finds out from the designer the number of new functions the designer intends to add on to the list. For each function, the name of the function, the status, the frequency of the function, the number of entities accessed, the path chosen to select this entity and the volume of records to be accessed in each transaction are recorded. The program checks if the named attribute or the relationship exist. In the situation when a named attribute or relationship does not exist, the program informs the designer and gives the designer an opportunity to correct the error. Once the details of all the new functions are recorded the program writes back all the function details in the file named 'hospfunc' and 'nooffunc' is accordingly updated.

The information collected is represented in a tabular form (Figure 3.3). The details about the functions of the hospital information system stored by this program are in Appendix E.

FUNCTION-NAME:	FREQUENCY:	RESPONSE-TIME:	STATUS:
ENTITIES ACCESSED	SELECTION - CRITERIA	RANGE/EQUALITY	
PATIENT	pat-name	EQUALITY	
CLINICAL TIME SCHEDULE	APPOINTMENT		
CLINICAL SESSION	clinic-no	EQUALITY	

Figure 3.3 Functional Analysis Table (1)

3.4 Analysing usage pattern

A second program, 'ANALYSIS.P', analyses the information recorded by 'FUNCTEST.P' and produces tables which give consolidated views of the usage pattern of the entities, their attributes and the relationships.

The table, 'eusagemat', denotes for each entity which attributes are used most often to access that entity. The table, 'eusagemat', consists of an array of records named 'eusagedet'. The record, 'eusagedet', in turn is an array of records named 'analdet'. The record, 'analdet', has two components, 'primdet' and 'seconddet'. They denote how attributes are used in primary and secondary functions respectively. Both 'primdet' and 'seconddet' have a record called 'accdet' as component. It specifies the frequencies of an attribute being used in an equality clause and as a range access. An equality clause means that the value of the attribute provided for searching is compared with the value of the attribute of the required entity to give an exact match. For a range access there is no need for an exact match but the value of the attribute of the required entity should be in the range provided by the search clause. Therefore, each attribute has four types of usage frequency. They are:

1. Primary functions - equality clause
2. Primary functions - range clause
3. Secondary functions - equality clause
4. Secondary functions - range clause.

The program, 'ANALYSIS.P', also specifies the total frequency of relationship usage with regard to the access of an entity from another entity, and is represented by the table 'rusagemat'. The table, 'rusagemat', consists of an array of records named 'rusagedet'. The record, 'rusagedet' has two components 'primfreq' and 'secfreq'. Both 'primfreq' and 'secfreq' are represented as integers, and denote the frequency of usage of the relationships in primary and secondary functions respectively.

The entities, relationships and functions are read in from respective lists created by the designer. For each function, it is noted how a particular entity is being accessed. If it is being accessed by way of its attributes or its primary key, then it is noted whether it is being used in a range clause or equality clause. Depending on that, the frequency of the function is added on to the appropriate cumulative usage frequency mentioned earlier. If the entity is being accessed by means of a relationship, then the frequency of the function is added to the primary accumulative usage frequency or the secondary usage frequency, depending on the status of the function.

The result of the analysis is represented in tabular form (Figures 3.4 and 3.5). The output from this program for the hospital information system can be found in Appendix E.

3.5 Graphical representation of the data

The entity model can be represented graphically (Figure 3.6) where every node of the graph represents an entity and the edges represent either an attribute or a relationship. Edges between two nodes represent a relationship or an access path to travel from one entity to the other. Edges with only a single node on one side represent attributes. These are used as access paths to entities.

An inward edge, i.e. an edge marked with an arrow pointing towards the entity, denotes that the entity towards which the arrow points is being accessed through that relationship, when the value of the entity occurrence on the other side is known. The edges, therefore, show the navigational paths. The total traversal frequency of each of these paths is calculated. Details about whether the attributes or relationships are being used to retrieve a single record or a subset of records are also noted. Figure 3.6 shows this graphical representation of part of the data.

ENTITY NAME : PATIENT				
ATTRIBUTE	PRIMARY		SECONDARY	
	RANGE FREQUENCY (times/day)	EQUALITY FREQUENCY (times/day)	RANGE FREQUENCY (times/day)	EQUALITY FREQUENCY (times/day
date-of-birth	200	0	0	50
pat-no	0	250	150	0

Figure 3.4 Entity-Attribute Usage Table

RELATIONSHIP NAME	PRIMARY		SECONDARY	
	RANGE FREQUENCY (times/day)	EQUALITY FREQUENCY (times/day)	RANGE FREQUENCY (times/day)	EQUALITY FREQUENCY (times/day)
APPOINTMENT	500	0	100	0
PAT-CONS	300	100	500	0

Figure 3.5 Relationship Usage Table

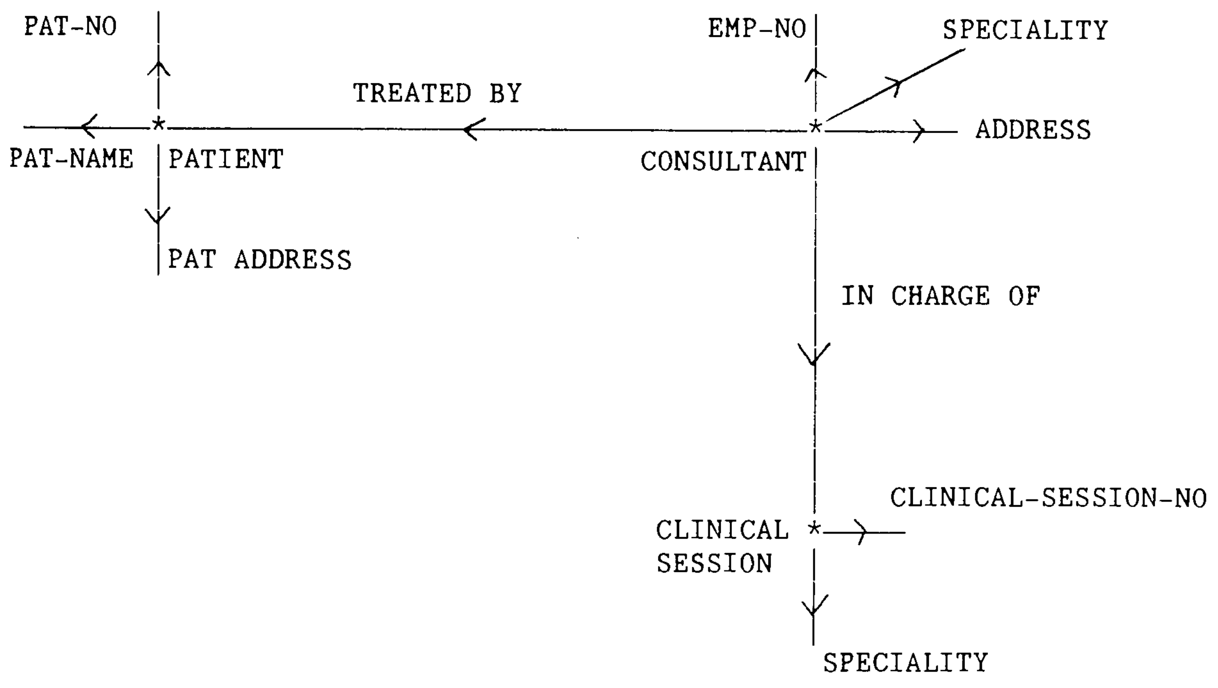


Figure 3.6 Graphical Representation of Data

3.6 Conclusion

In the physical design phase, information about usage pattern is used to decide on the structure of the stored files. Of course, consideration has to be given to the importance of the applications. The objective of functional analysis is to give enough information about the functions, so as to establish the way data is to be stored in the data base in order to fulfil the primary functions efficiently. Secondary applications are satisfied by selecting additional record access paths if necessary. In this phase the basic functions of the organisation are identified, the degree to which the functions share data is indicated, and this information forms the basis for the physical design phase.

CHAPTER 4

LOGICAL DESIGN

4.1 Introduction

In the past, the methods used in the design of data models for data base management systems (DBMS) implementation have largely been based on trial-and-error. This approach to design leads to solutions that do not necessarily meet the users' requirements. The success of a three-level DBMS architecture depends on how well the structure of the organisation is analysed and how efficiently this structure is implemented on the target DBMS.

There are two mapping processes in a data base design. First, the conceptual model (or the integrated users' views) is mapped on to the data model of the target system. This is referred to as the logical design. This uses the conceptual model and restructures it into the logical structure of the target system. This logical model provides a non-redundant (no duplication of entity or relationship) and usage independent (not dependent on how the data is being used) base from which the internal and external models are mapped. The accuracy of the conversion of the conceptual model to the logical model of the target system is crucial to the success of data base implementation. The next step is the physical design. In this phase the storage level representation of the data of the target system is dealt with, the alternatives for the physical implementation are considered and the best possible storage structure is chosen (Chapter 7). In this chapter, the issues concerning the logical design step will be discussed.

In general, we should be able to map the conceptual model on to any data model as dictated by the target system. The mapping should be such that there is no loss of information, the original processing specifications are satisfied and it is independent of any physical implementation.

4.2 Logical mapping

Logical mapping is, effectively, a process of converting the syntax of the conceptual model to the syntax of the logical model for the target DBMS. The data structure of a DBMS is reflected in the structuring elements available for defining the logical model of that DBMS. More often than not, there will be some aspects of the conceptual model which cannot be represented by the structural elements of the DBMS. In such cases, it will be necessary to modify the structure of the conceptual model so that it is possible for the logical model of the target DBMS to capture the information content of the conceptual model.

During the process of conversion it is crucial to observe that greater data redundancy is not introduced and maintenance of the logical structure does not become complex. Logical data base design may involve some measure of compromise between introducing complexity or losing information, depending on the richness of the data structure of the target DBMS.

The logical model of a data base will reflect the structure of the information content of the data base. It is essential to maintain the integrity of the information. Therefore, for logical model design we need to know the logical structure of the information content of the conceptual model and the information necessary to ensure the integrity of this information. To maintain the integrity of the information in the data base it is important to minimise the side effects of updating the logical structure of the data base. In chapter 2, the structure of the conceptual model was discussed. The following sections describe the types of update operations that can be performed on the conceptual model and the effects they have on the model.

4.2.1 Update operations

In terms of an E-R conceptual model there can be three types of update operation:

1. Inserting or deleting an entity occurrence
2. Inserting or deleting a relationship instance
3. Modifying the values of the attributes of an existing entity or relationship.

These update operations should not leave the logical structure in an inconsistent state, that is, the update operations should not introduce any discrepancies in the structure of the data. For example, there should not be any entity occurrence participating in a relationship instance when the entity occurrence has been deleted by an update operation. To maintain a consistent logical structure, further update operations are needed. They are the side effects of an update operation. The side effects should be limited, obvious and controllable.

4.2.2 Effects of the update operations

The update operations in an E-R conceptual model can induce the following effects.

Deleting an entity occurrence 'e' causes:

a) the deletion of an instance of a relationship in which 'e' participates, and

b) the deletion of one or more entities in the domain of a relationship, with 'e' in the range where the membership of the domain is obligatory.

In Figure 4.1, (a hypothetical example used for illustrative purposes) if the information about a patient M SMITH is deleted from the data base, the relationship instance which implies that M SMITH is being treated by DR WILLIAMS should also be deleted. However, in the case of the deletion of a consultant entity occurrence DR COX, it is not sufficient to delete only the relationship instances in which DR COX participates; it is also necessary to delete the information about the patients who are being treated by DR COX. As the membership of the 'PATIENTS' entity is obligatory in this relationship, it is essential that the data base only holds information about those patients who are currently in the care of a consultant. The circumstances would be different if it were possible for a patient to be treated by more than one consultant. In such a case, it would not always be necessary to delete the information about the patients being treated by the consultant because they can still be in another consultant's care.

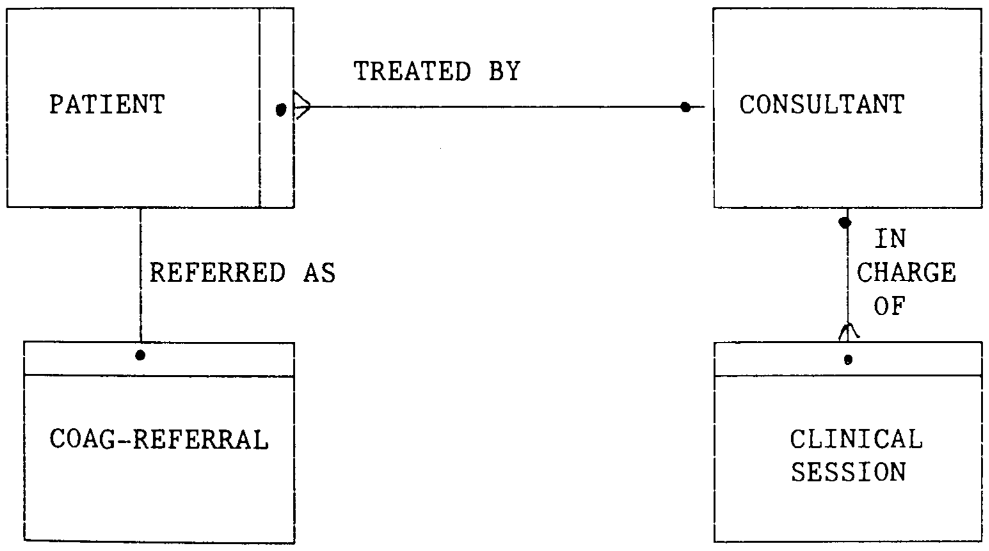


Figure 4.1 Conceptual Model

Deleting a relationship instance 'Ex → Ey' causes:

a) the deletion of the entity occurrence 'Ey' if the degree of the relationship 'Ex → Ey' is 1:1 or 1:n and the membership of 'Ey' is obligatory, or

b) the deletion of the entity occurrence 'Ex' if the degree of the relationship is 1:1 and the membership of 'Ex' is obligatory, or

c) the deletion of the entity occurrences 'Ex' and 'Ey' if the degree of the relationship is 1:1 and the membership of 'Ex' and 'Ey' are obligatory.

Following Figure 4.1, if an instance of the relationship 'CONSULTANT → CLINICAL SESSION', of degree 1:n, is deleted, then it will be necessary to delete the information about the entity 'CLINICAL SESSION' taking part in this relationship instance. Information about GYNAE-CLINIC-3 has significance in the hospital data base only as long as DR COX is in charge of this clinic. As a clinical session can only be in the charge of a single consultant, any information regarding this session is of no importance unless a consultant takes charge of it. In the same example, 'COAG-REFERRAL' refers to a single patient. It has no significance unless it relates to a patient. If the information that coag-referral C.R.123 refers to the patient D SMITH is deleted then any detail about C.R.123 is irrelevant for the hospital data base.

Inserting an entity 'Ex' needs:

a) any entity 'Ey' in the range of a relationship 'Ex → Ey' with 'Ex' in the domain, to exist already if the membership of the domain is obligatory.

In the example 4.1, if we insert in the data base any information about the coag-refferal C.R.123, the information about the patient D SMITH should already exist. Similarly, before we can introduce the entity GYNAE-CLINIC-3 in the data base, the information about Dr COX, who is in charge of this clinical-session, should be present in the data base.

Inserting a relationship instance 'Ex → Ey' requires:

a) that the entity in the domain 'Ex' and the range 'Ey' to exist already.

If in the example 4.1 we wish to specify details about a 'PATIENT', such as patient M RAY having diabetes, the information about entity occurrence (M RAY) and entity 'DISEASE' (DIABETES) should already exist in the data base.

4.3 Conclusion

Our approach to logical data base design is to formulate rules for transferring the E-R model into relational, CODASYL or other models, whilst preserving the atomicity of update operations and controlling side effects. An atomic update operation is one where a single update operation affects only a single logical object. Hence, it is desirable that information about a single object is kept in a single place in the logical model. To control the side effects of an update operation, the induced update operations should be similar to those described for the E-R model.

This phase is a logical process and rules can be formulated if the data structures of the target DBMS are known. In the following chapters, rules for mapping an E-R model to a relational data model and a CODASYL data model are described.

CHAPTER 5

RELATIONAL MAPPING

5.1 Introduction

In this chapter, rules for mapping the E-R conceptual model on to the logical model of a relational DBMS will be described. Before the mapping rules are considered, the structure of the relational model will be discussed.

5.2 Characteristics of the relational model

In the relational model data is organised in a tabular form. Rows of these tables are known as tuples and columns are known as attributes. A domain is the collection of values from which the values of the attributes are drawn. A single attribute or a combination of attributes which identify a tuple uniquely is known as the primary key. When the primary key is made of a combination of attributes it is known as a composite key. Sometimes there is more than one attribute or a combination of attributes which has the property of identifying a tuple uniquely. They are called candidate keys. An attribute or a composite attribute of a relation is called a foreign key if it is the primary key of another relation.

To use the relations as a time-varying representation of data, we should be able to insert, delete and modify the tuples. The update operations may sometimes yield uncontrollable side effects. To keep these side effects to a minimum we should use the relations in their normalised form.

5.3 Normalisation

Date (1986) discusses functional dependence which is the crucial concept on which the theory of normalisation is based. An attribute 'Y' of relation 'R' is functionally dependent on attribute 'X' of 'R' if and only if each 'X' value in 'R' has associated with it precisely one 'Y' value in 'R'. The concept of functional dependence can be extended to cover the case where 'X' or 'Y' may be a combination of attributes. 'Y' is fully functionally dependent on 'X', where 'X' is a combination of attributes, if 'Y' is functionally dependent on 'X' but not on any subset of 'X'.

Date (1986) defines four normal forms as follows:

'A relation R is in FIRST NORMAL FORM (1NF) if and only if all underlying domains contain atomic values only.

A relation R is in SECOND NORMAL FORM (2NF) if it is in 1NF and every nonkey attribute is fully dependent on the primary key.

A relation R is in THIRD NORMAL FORM (3NF) if it is in 2NF and every nonkey attribute is nontransitively dependent on the primary key.

A relation R is in FOURTH NORMAL FORM (4NF) if and only if, whenever there exists a multivalued dependency in R, say of attribute B on attribute A, then all attributes of R are also functionally dependent on A.'

It can be stated that a relation is said to be in the first normal form if, and only if, all the attributes contain non-decomposable values. A relation is in second normal form if it is in first normal form and there are no nonkey attributes which are functionally dependent on only part of the key. A second normal form relation is in third normal form, if all the nonkey attributes are nontransitively dependent on the primary key and not dependent on any other nonkey attributes.

Ullman (1982) also argues that normalised (3NF) relations solve the problems of update anomalies. Boyce-Codd normal form is stronger than third normal form in the elimination of update anomalies. Ullman (1982) defines Boyce-Codd normal form:

'A relation scheme R with dependencies F is said to be in Boyce-Codd normal form if, whenever X → T holds in R, and A is not in X, then X is a superkey for R; that is, X is or contains a key'

However, Ullman points out that Boyce-Codd normal form is too constraining and it is not necessary to reduce relations to this form for most practical applications of data bases. Third normal form is adequate for most cases.

The concept of <u>multivalued dependency</u> needs to be mentioned before the fourth normal form relations are illustrated. An attribute 'B' of 'R' is said to have a multivalued dependence on an attribute 'A' of 'R' if for a given 'A' there is not a single 'B' but a well defined set of 'B'. Functional dependence is just a special case of multivalued dependence. A relation is in fourth normal form, if it is in third normal form and if there exists a multivalued dependence (say, 'B' of 'R' on 'A' of 'R'). All the other attributes are fully functionally dependent on 'A'.

Whilst 3NF relations are adequate for most practical applications, they suffer from certain inadequacies. 3NF relations which have more than one candidate key, where the candidate keys are composite and have at least one attribute in common, are not free of update anomalies. If our objective is to keep redundancy and side effects of the update operations to the minimum, then it is desirable to keep the relations in a relational model in this fully normalised form (4NF).

5.4 Relational design

Our approach to relational logical model design is to convert the E-R conceptual model to a relational model without losing any semantic information. The other objective is to minimise the uncontrollable side effects caused by any update operations. One should be able to create or destroy all the properties of an entity or relationship occurrence at the same time as the creation or destruction of the entity or relationship occurrence. This may be achievable through the creation or destruction of a single tuple in the relational model. Moreover, a single relational update should be sufficient to update the property value of an entity or a relationship.

The identifier of an entity set is the property which uniquely identifies an entity occurrence. The identifier of a relationship is the combination of the identifiers of the entities taking part in the relationship. To avoid uncontrollable side effects each entity or relationship occurrence must be represented by a unique identifier all through the relational model. Undesirable side effects can also be eliminated if each relation in the relational model contains information about only a single entity or relationship. This ensures that the deletion of an entity or relationship does not delete any other entities or relationships.

Thus, it can be stated that there should be a relation in the relational model corresponding to each entity set. The identifier forms the primary key of that relation. The attributes of a relation are the properties of the entity. Each tuple corresponds to each entity occurrence. Corresponding to each relationship in the conceptual model, there should be a relation in the relational model. The primary key of the relation is the identifier of the relationship. Each tuple in the relation represents an instance of the relationship. We can have a few exceptions to these mappings. These, we hope, will improve the efficiency of the relational model without the loss of any semantic information.

If two entities 'Ex' and 'Ey' take part in a 1:1 or 1:n relation 'Rxy' where the membership of 'Ey' is obligatory, it is not necessary to write a relation representing the relationship 'Rxy'.

Instead, the identifier of the non-obligatory member 'Ex' can be posted to the relation table for entity 'Ey' as an attribute. For the entity 'Ey' to exist it has to take part in this relationship. Thus, it can be assumed that taking part in this relationship is a property for this entity type. To delete an instance of the relationship 'Rxy', it would be necessary to delete only that entity occurrence of 'Ey'. This would automatically delete the relationship and the entity occurrence as required.

In some cases, the identifier of an entity type forms a part of the identifier of another closely associated entity type. In such cases, it is not necessary to have a relation representing the relationship between these two entity types, because the information is already available from the primary key of the second entity type. This is just a special case of the obligatory membership of an entity discussed in the last paragraph. The membership of the entity, with the borrowed identifier, will be obligatory in the relationship between this entity and the entity from which it borrowed the identifier.

In a 1:1 or 1:n relation 'Rxy' between the entities 'Ex' and 'Ey' where the membership of 'Ey' is obligatory the relationship properties are considered the properties of the entity 'Ey'. As the membership of 'Ey' is obligatory, for each instance of 'Rxy' there will be a corresponding occurrence of 'Ey' and vice versa. Hence, the attributes of the relationship instance can be considered to be the attributes of the entity occurrence of type

'Ey'. If the membership of 'Ey' is non-obligatory the relationship might possess attributes. A relationship 'Rxy' of degree m:n, can also have some properties associated with it. It does not matter whether the membership of the entities 'Ex' and 'Ey' taking part in it are obligatory or not. These attributes cannot be considered as a part of the entity attributes because these properties only describe the specific relationship. These properties do not describe the entities when they are not taking part in that specific relationship. When mapping to a relation in the relational model, m:n relationships with or without properties can be treated alike.

5.4.1 Formal mapping rules

From the above discussions we can formulate the following mapping rules.

1. Identify each entity set by a unique attribute throughout the relational model.

2. For each entity set 'Ex' form a relation 'R(Ex)'. The primary key of the relation is the key attribute, that is, the identifier of 'Ex'. The attributes of the relation are (i) the properties of the entity 'Ex', (ii) the identifier of the entity 'Ey', if and only if there is a 1:1 or 1:n relationship 'Ey → Ex' where the membership of 'Ex' is obligatory and the identifier of 'Ey' does not form a part of the identifier of 'Ex'.

3. For each relationship 'Rxy' between the two entities 'Ex' and 'Ey' of degree 1:1 where the membership of both the entities are non-obligatory define a relation 'R(Rxy)'. The attributes of 'R(Rxy)' are made up of the identifiers of 'Ex' and 'Ey' and the attributes of the relationships. The identifiers of 'Ex' and 'Ey' are both candidate keys.

4. For each relationship 'Rxy' between the two entities 'Ex' and 'Ey' of degree 1:n where the membership of both the entities are non-obligatory define a relation 'R(Rxy)'. The attributes of 'R(Rxy)' are made up of the identifiers of 'Ex' and 'Ey' and the attributes of the relationship. The primary key of 'Rxy' is the identifier of 'Ey'.

5. For each relationship 'Rxy', between the entities 'Ex' and 'Ey' of degree m:n, or between more than two entities 'Ex', 'Ey',......'En', where the membership of the entities may be obligatory or non-obligatory define a relation 'R(Rxy)'. The attributes of the relation are the identifiers of the participating entities and the properties of the relationship 'Rxy'. The primary key of the relation 'R(Rxy)' is the combination of the identifiers of the entities taking part.

5.4.2 The mapping program

Program, 'RELMAP.P', examines the list of entities and relationships and applies the mapping rules to obtain the resultant relational logical model. The output from the program is a list of normalised relations.

The record, 'norment', represents an entity in the relational model and 'normrel' represents a relationship in a relational model. The record, 'norment', has four components and they are:

nename	name of the entity
noofprops	total number of attributes
identcnt	number of attributes in the identifier
norentatts	array of a record named props.

The record, 'props', has two integer components. The first one pointing to an entity in the entity list, the second one pointing to an attribute of that entity.

The record, 'normrel', has the following components:

nrname	name of the relationship
identcnt	number of attributes in the identifier
norrelidnt	array of 'props' identifying the attributes in the identifier
norkeycnt	number of attributes which is actually equal to the keycount of a participating entity but these attributes are not a part of the key of relation
norrelkey	array of 'props' identifying the attributes mentioned in the description of norkeycnt
nrattcnt	number of attributes of the relationship

nrelatt	pointers pointing to the attributes of the relationship.

The variable, 'norentchtr', is an array of 'norment' representing the list of normalised entities and 'norrelchtr' is an array of 'normrel' representing the list of normalised relationships. First the entities and relationships are read from the list. Each relationship is examined. Rules defined in the previous section are applied to these relationships.

If the relationship between 'entitya' and 'entityb' is of degree 1:1 or 1:n and the membership of 'entityb' is obligatory then no relation corresponding to this relationship is created. Instead, the identifier of 'entitya' is added on as an attribute of 'entityb'. Whereas, if the membership of 'entityb' is non-obligatory then a relation corresponding to this relationship is created. If the relationship is of degree m:n, a relation corresponding to the relationship is created independent of the membership classes of the participating entities.

When these mapping rules are applied to the example (Figure 5.1), the resulting relations are shown in Figure 5.2.

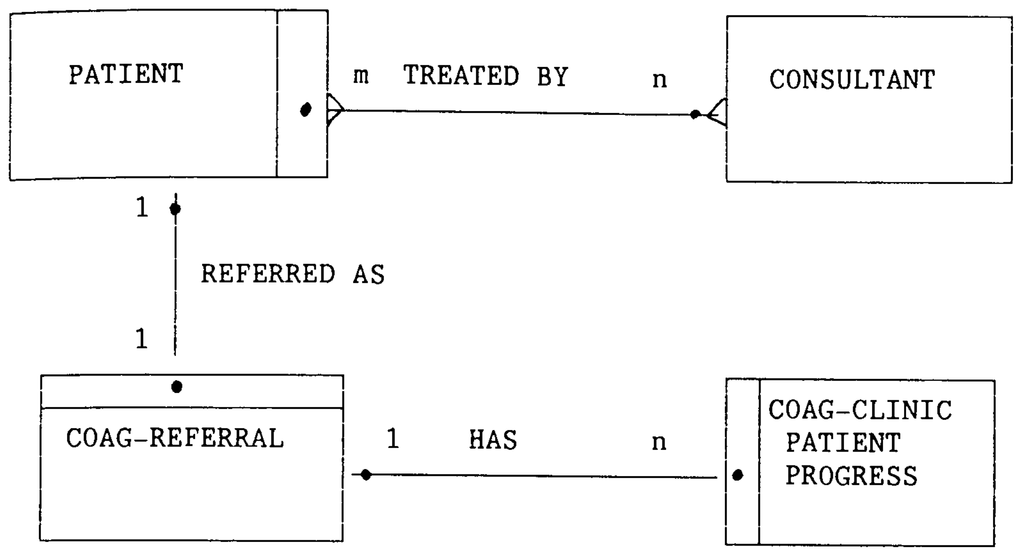


Figure 5.1 Entity Relationship Model

PATIENT (pat-no, pat-name, address,..........)

CONSULTANT (emp-no, name, address, speciality,....)

CONS-PAT (emp-no, pat-no,......................)

COAG-REFERRAL(coag-ref, drug-regime, therapy,
pat-no......)

COAG-CLINIC-PATIENT-PROGRESS (coag-ref, date, result.....)

Figure 5.2 Relational Model

5.4.3 Resultant relations remain normalised

The identifiers and the properties of the entities and the relationships are defined over atomic or non-decomposable value sets. The relations derived from them would also be defined on atomic domains. Hence, the relations are in the first normal form.

The relations obtained from any entity set 'Ex' by applying the second mapping rule would contain the identifier 'IDEx', 'PEx' the properties of 'Ex' and the identifier of the entity 'Ey', 'IDEy', if there exists a 1:1 or 1:n relationship 'Ex → Ey' and the membership of 'Ex' is obligatory. The functional dependencies in the relations are in the form 'IDEx' → 'PEx', 'IDEx' → 'IDEy' and the nonkey attributes like 'PEx' and 'IDEy' are directly fully dependent on the primary key 'IDEx'. There is no multivalued dependence. Hence the relations are in the fourth normal form.

The form of dependencies in the relations derived by applying the third mapping rule for a 1:1 relationship between 'Ex' and 'Ey', where the membership of both the entities are non-obligatory, is in the form 'IDEx' → 'IDEy' and 'IDEx' → 'PRx'. 'IDEx' and 'IDEy' are nontransitively and fully dependent on each other. 'PRx' is also fully dependent on 'IDEx' and 'IDEy'. 'IDEx' and 'IDEy' are both candidate keys, hence the relations are in the fourth normal form.

The relations derived by applying the fourth rule of mapping for 1:n relationships between entities 'Ex' and 'Ey', where the membership of both the entities are non-obligatory, will have functional dependence of the form 'IDEy' → 'IDEx' and 'IDEy' → 'PRx'. This will again be in fourth normal form.

The relations obtained by applying the fifth rule are defined over the key attributes of the entities taking part and the properties of the relationships. The form of the dependency is 'IDEx', 'IDEy',....,'IDEn' → 'PRx'. There is no functional dependency among the identifiers of the entities taking part and 'IDEx', 'IDEy',....,'IDEn' form the primary key. The attributes are functionally dependent on the primary key. The relations cannot be decomposed into other third normal form relations because the relationships these represent are non-decomposable. Hence, they are in fourth normal form.

5.5 Conclusion

The mapping rules were designed so that the derived relations had no possibilities for transitive, partial or multivalued dependencies. Thus, they are all in the fourth normal form. The rules were formulated to control the side effects, that is, to control the update anomalies. This correspondence justifies the view that fourth normal form relational models are desirable because they keep the uncontrollable side effects of update operations to a minimum. The crucial factor for a relational model design is that one relation should contain information about only one logical object. Violation of this will produce violation of the normal forms. A model so designed would suffer from undesirable update anomalies.

CHAPTER 6

CODASYL MAPPING

6.1 Introduction

In this chapter rules for mapping the E-R conceptual model into the logical model of a CODASYL type of DBMS will be described. Before the mapping rules can be defined, it is necessary to describe the structure of CODASYL DBMS.

6.2 The CODASYL structure

In CODASYL (1978) the term schema can refer both to logical model and conceptual model. It may be helpful to bear in mind that schema is synonymous with the logical view appropriate to a particular information system. A CODASYL logical model described by Olle (1978) includes many aspects which are not relevant to the logical view of the data, but in fact describes the physical properties of the data. For our purpose we will choose only those aspects of the logical model which are necessary for the logical data base design.

The two basic elements of the CODASYL structure are records and sets. A record type is the collection of related data about a particular object type. A set type is an association between two or more record types. In a hospital data base 'CONSULTANT', 'NURSE' and 'PATIENT' are examples of records. The relationship 'TREATED BY', which shows the association between record types 'CONSULTANT' and 'PATIENT', is an example of a set type. Each set type is owned by an owner member record type and has one or many member record types. In the above example of a set type, 'CONSULTANT' is the owner and 'PATIENT' is the member record type. A uniquely identified record from a record type is known as a record occurrence and a uniquely identified set from a set type is termed a set occurrence.

A set type represents a relationship of degree 1:1 or 1:n between the owner and member record types. A set type exists to provide access from one record to another. The rules that apply concerning the basic structure of records and sets are :

1. A record type can be an owner in one set type and member in another set type.

   As for example (Figure 2.4), 'CLINICAL SESSION' is the owner in the set which shows association between 'CLINICAL SESSION' and 'CLINICAL TIME SCHEDULE' but a member in the set which shows the association between 'CONSULTANT' and 'CLINICAL SESSION'.

2. A record type can be member or owner in more than one set type.

   In the hospital information system (Figure 2.4), 'CONSULTANT' is the owner of sets which show the association between 'CONSULTANT' and 'CLINICAL SESSION' and 'CONSULTANT' and 'SURGICAL SESSION'.

3. There may be more than one set type defined over the same two record types.

   In the hospital information information system, there are two set types 'IN CHARGE OF' and 'ON CALL FOR' defined over the record types 'CONSULTANT' and 'WARD'.

4. There is no limit to the number of record or set types.

5. Cyclic structuring is allowed.

   Figure 6.1 shows the cyclic structuring between 'PATIENT', 'REQUESTS FOR BLOOD' and 'BLOOD RESERVE'.

For every record it is necessary to define and describe each field.

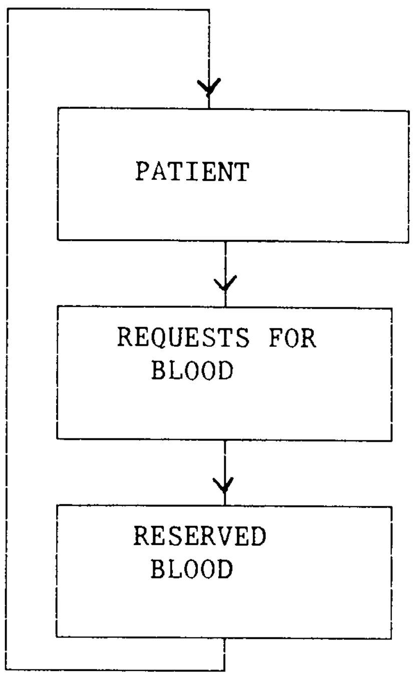


Figure 6.1 Cyclic Structuring

6.2.1 Location mode

The location mode specifies how the DBMS stores and accesses each record. The location modes available in a CODASYL DBMS are as follows:

1. DIRECT: In the direct mode the user is allowed to pass the data base key (key used to physically locate a record) to the system. In this mode a record is stored quickly but there is no specific way of retrieving it.

2. CALC: In this mode the DBMS uses the record key to determine the data base key. It is not specified whether a randomising or an indexing technique is used.

3. VIA-SET: In this mode the record is stored physically close to the other members of the set type named.

4. SYSTEM: In this mode, it is possible to access and store the record efficiently without using the data base key.

6.2.2 Order and removal/storage class

In the logical model description, apart from defining the owner and member records of the set occurrences, the ORDER and the REMOVAL/STORAGE CLASS of the member records of the set types have to be described. The ORDER shows where a new member occurrence should be inserted. The options that are available in the CODASYL DBMS are the following:

FIRST-	the member is inserted immediately following the owner.
LAST-	the member is inserted immediately preceding the owner.
NEXT-	the member is inserted immediately following the last member occurrence accessed.
PRIOR-	the member is inserted immediately preceding the last accessed record in the set.
IMMATERIAL-	the member is inserted at the convenience of the DBMS.
SORTED-	the member is inserted in a position so that some declared ordering on the members is preserved.

The STORAGE class specifies whether a record, when it is first created, is attached to a set or not. Two types of STORAGE class are possible. When a record is added to the data base for the first time, it is built up in a special area in core and then a 'store' is executed on that record. After the store has been executed, the DBMS checks all the set types in which the record type participates. If the STORAGE class is MANUAL then the new record is not connected to any set occurrence. If the storage class is AUTOMATIC, then it is connected to one set occurrence of the set type. REMOVAL class describes what happens when a record is deleted from the data base. FIXED refers to an option that can be taken to describe what is allowed to happen to the member record once it is inserted into an occurrence of the set. FIXED means that the member cannot be removed from a particular set unless it is deleted from the data base. If the REMOVAL class is defined as MANDATORY a record occurrence can be removed from one set occurrence to another set occurrence of the same type or deleted. If the REMOVAL class is declared as OPTIONAL then the member record occurrence can be removed from the set occurrence entirely and remain unconnected to the set type.

6.3 CODASYL mapping rules

Our objective in this section is to define rules to map the E-R model on to a CODASYL logical model. The CODASYL (1971/78) logical model definition includes many of the physical design aspects. For our purpose we will choose only those aspects which are concerned with the logical model design because physical design aspects are covered by the particular implementation and should be separate from the logical description. These are the specification of record and set types, data items, keys and set removal/storage classes.

The guidelines which were followed for the relational mapping apply also to the CODASYL mapping. The mapping from the E-R model to the CODASYL structure should be such that no information is lost in the process. All information related to an entity or relationship should be kept together, so that they can be created or destroyed by one CODASYL instruction. Moreover, the definition of the STORAGE/REMOVAL class of the members of a set type should be used to express the membership of the entities taking part in a relationship.

Inspecting the E-R model and the CODASYL logical model it can be seen that, for every entity in the E-R model there should be a corresponding record construct in the CODASYL model. Though the set construct corresponds to the relationship concept in the E-R model, some of the relationships need modification before such a mapping can be performed. There are two reasons for such modification. Firstly, the relationship in the E-R model can be of degree 1:1, 1:n or m:n. The set construct in the CODASYL model supports only relations of degree 1:1 and 1:n. Hence, the m:n relationships in the E-R model need some changes before we can map them on to a CODASYL set. Secondly, the relationships in the E-R model may have some attributes associated with them. The set in the CODASYL model is used as a navigational tool to access a record from another record. It is not possible to associate any attributes with it. Therefore, all the relationships in the E-R model which have some attributes associated with them should be modified before they can be mapped on to the CODASYL structure.

6.3.1 Modification of relationships

To modify an m:n relationship between two entities to 1:n structures, in order to conform to the CODASYL model, a new entity is created. This entity has the identifier of the relationship as its identifier. The original many-to-many relationship can now be replaced by two one-to-many relationships between the original entities and the newly created entity. For example, there is an m:n relationship between 'PATIENT' and 'CLINICAL SESSION'. This can be represented by two relationships 'PATIENT AND PATIENT-PROGRESS' and 'CLINICAL SESSION AND PATIENT-PROGRESS', where 'PATIENT-PROGRESS' is the third entity which has been created.

Similarly, for every relationship with attributes, a new entity is created. The identifier of the relationship forms the primary key of the entity and the properties of the relationship form the attributes of the entity.

Before we can state the degree and the membership of the relationship between the new entity and the original entities, it is necessary to specify the cases where a relationship might possess some attributes. Any relationship 'X→Y' of degree 1:1 or 1:n where the membership of 'Y' is non-obligatory, may have some attributes associated with it. If the membership of 'Y' is obligatory in the relationship, the properties can be assigned to the entity 'Y'. If 'X→Y' is of degree m:n the relationship might have some attributes, whether the membership of 'X' or 'Y' is obligatory or not. Say 'Z' is the new entity created instead of the relationship 'X→Y' there will be two new relationships, one between 'X' and 'Z' and the other between 'Y' and 'Z'. The degree of the relationships 'X→Z' and 'Y→Z' will depend on the degree of the original relationship 'X→Y'. If the degree of 'X→Y' was 1:1 then both 'X→Z' and 'Y→Z' will be of degree 1:1. However if the degree of 'X→Y' was 1:n, then 'X→Z' will be of degree 1:n and 'Y→Z' will be of degree 1:1. If the original relationship was of degree m:n then both the newly created relationships would be of degree 1:n. The membership of the entity 'Z' in the relationships 'X→Z' and 'Y→Z' will be obligatory in all the above cases. Once these modifications are performed on the E-R conceptual model, the resultant E-R model will only have relationships of degree 1:1 and 1:n. The relationships will not have any properties associated with them. This modified E-R model can now be easily mapped on to the CODASYL structure. For every entity we can have a record, and for every relationship we can define a set.

6.3.2 Choice of storage/removal class

The concept of the storage and removal class can also be used in the logical mapping. If the membership of an entity is obligatory in a relationship, then it must participate in that relationship when it is created. The same applies if we assign storage class 'automatic' to the member record of a set. If the membership is non-obligatory then it is enough to assign storage class 'manual' to the member record of a set. Assignment of the removal class is not as straightforward as the assignment of storage class. The information available from the E-R model is not enough to decide whether a removal class of fixed or mandatory is to be assigned. Before we can make these decisions, the entity life-cycle has to be analysed or more information needs to be added to the membership definition of the E-R model. The entities which are created by our modification process will not only have obligatory membership but also they cannot enter into the same relationship with other entities of the same type. Hence, we can assign these member records the removal class of fixed. Any other entities with obligatory membership in a relationship can be assigned a mandatory removal class and the entities with non-obligatory membership can be assigned an optional removal class.

The choice of the storage/removal class will control the side effects of the design model. In a relationship 'X→Y', of degree 1:1 or 1:n, if the membership of 'Y' is obligatory, and 'X' is deleted from the data base, then it is crucial that 'Y' is erased as well. Choosing the option mandatory/automatic for these types of relationships will guarantee that when 'X' is deleted 'Y' will automatically be deleted. These deletions can propagate more deletions. The automatic option for the storage class also guarantees that when the member record is created it will be connected to the appropriate set. Hence, when a new entity is created, the DBMS will automatically establish the connection between this entity and other entities through the relationships in which the membership of this entity is obligatory. Assigning the option optional/manual to a member record, ensures that the record can exist independently in the data base, even if it is not a member of that set type. Entities with non-obligatory membership can be assigned storage class 'manual'. Generally, it can be said that we can assign a removal class 'optional' to an entity with non-obligatory membership. There might be situations where an entity may have a non-obligatory membership, but once it enters into a relationship with another entity, it might not be possible to destroy the relationship, or change it to a relationship with another entity of the same type, without deleting the entity. In such cases, it will not be possible to assign a removal class of optional. This information is not available from an E-R model. Once the functional analysis is done, it will be necessary to scan the storage/removal class of the member records of the set and take appropriate action. Problems of this nature are bound to arise in CODASYL mapping due to the absence of a definite demarcation between the logical and physical model. At this stage, we can still define some rules to map the E-R data model on to the CODASYL structure.

6.4 Formal mapping rules

The mapping rules are as follows:-

1. For every relationship 'X→Y' of degree m:n in the E-R model, form a new entity 'Z' which has the identifier of the relationship as its identifier and the attributes of the relationships, if any, as its attributes. Instead of 'X→Y', set up two relationships 'X→Z' and 'Y→Z' of degree 1:n. Assign membership class obligatory to 'Z' in both these relationships.

2. For every relationship 'X→Y' of degree 1:1 which has some attributes, form an entity 'Z' which has the relationship's identifier and attributes as its identifier and attributes respectively. Instead of 'X→Y' set up two relationships 'X→Z' and 'Y→Z' of degree 1:1. Assign membership class obligatory to 'Z' in both the relationships.

3. For every relationship 'X→Y' of degree 1:n, which has some attributes associated with it, form an entity 'Z' which has the identifier and the attributes of the relationship as its identifier and attributes. Instead of 'X→Y', set up two relationships 'X→Z' and 'Y→Z' of degree 1:n and 1:1 respectively. Assign membership class obligatory to 'Z' in both these relationships.

4. For every entity set 'X', new or old, define a record R(X), the data items of which are formed from the identifier and attributes of the entity.

5. For each relation R(X→Y), where 'X' and 'Y' are genuine entity types, define a set type S(R) with R(X) as the owner and R(Y) as the member record type. Assign storage/removal class of manual/optional if the membership of the member entity is non-obligatory and assign membership class automatic/mandatory if the membership of the member entity is obligatory.

6. For each relation R(X→Y) where 'Y' is a modified entity, define a set type S(R), with R(X) as owner and R(Y) as the member record. Assign storage/removal class automatic/fixed to the member record type.

By applying the rules stated, we ensure that all information about an entity is kept together. The update operations described in chapter 4 can be performed by the data manipulation language operations STORE, ERASE and MODIFY. All the information about the relationships is kept either in a record which represents a modified relationship, or as a set structure. Figure 6.1 is the resultant CODASYL logical model obtained by applying these mapping rules to the example in Figure 5.1.

```
Record name is PATIENT
       pat-no               Pic 9(8)
       pat-name             Pic A(30)
       address              Pic X(30)

Record name is CONSULTANT
       emp-no               Pic X(6)
       name                 Pic A(30)
       address              Pic X(30)
       speciality           Pic A(8)

Record name is PAT-CONS
       emp-no               Pic X(6)
       pat-no               Pic 9(8)

Record name is COAG-REFERRAL
       coag-ref             Pic X(8)
       drug-regime          Pic X(15)
       commence-therapy     Pic x(30)

Record name is COAG-CLINIC-PATIENT-PROGRESS
       coag-ref             Pic X(8)
       date                 Pic 9(6)
       result               Pic X(40)

Set name is REFERRED AS
    Owner is PATIENT
    Member is COAG-REFERRAL Mandatory Automatic

Set name is HAS
    Owner is COAG-REFERRAL
    Member is COAG-CLINIC-PATIENT PROGRESS Mandatory Automatic

Set name is PATIENT-PAT-CONS
    Owner is PATIENT
    Member is PAT-CONS Fixed Automatic

Set name is CONSULTANT-PAT-CONS
    Owner is CONSULTANT
    Member is PAT-CONS Fixed Automatic
```

Figure 6.2 An Example CODASYL Logical Model

6.5 The mapping program

The program, 'CODMAP.P', maps the E-R model on to the logical data structure of a CODASYL type DBMS. The resultant records and sets are stored in the files 'codent' and 'codrel' respectively.

First the stored entities and relationships are read in from the list of entities and relationships stored by the designer. As all relationships cannot be directly mapped on to the CODASYL structure, they are modified to CODASYL compatible form. The list, 'modrelchart', holds all the modified relationships. It is represented by an array of records named 'modreln' which represents the relationship in the modified form and have the following components:

mrtype	a character denoting whether the relationship is original or modified
mrname	the name of the relationship
mentitya	pointer to the entitya
mdegenta	degree of entitya
mmembshipa	membership of entitya
mentityb	pointer to the entityb
mdegentb	degree of entityb
mmembshpb	membership of entityb

As the structure of the entities does not change there is no need to form a modified entity structure.

The program examines every relationship. If it is of degree 1:1 and 1:n and does not have attributes, it is not modified and the relationship is copied directly to the 'modrelchart'. If the relationship is of degree 1:n, has attributes and the membership of entityb is non-obligatory, then a new entity and two new relationships are created. The method adopted for this modification has been described earlier in section 6.3.1. The number of total entities is updated. The number of relationships is increased and the two new relationships are added to the 'modrelchart'. If the relationship is of degree m:n then again a new entity is created and two relationships are formed. The number of entities and relationships and the 'modrelchart' are updated.

The new entity set is now transferred to CODASYL record structure. The name of the entity forms the name of the record. The key of the record is formed from the primary key of the entity and the nonkey fields are formed from the nonkey attributes of the entity. The list of the records is written in the 'codent' file.

The modified set of relationships is transferred to the CODASYL set structure. The entitya becomes the owner record and the entityb becomes the member record. The membership class of the member record is derived from the membership class of the relationship. The method adopted for this derivation has been described earlier in section 6.3.2. The file, 'codrel', contains the list of sets.

6.5 Conclusion

The characteristics of a CODASYL structure are complex, but it is capable of representing the properties of our chosen data model. The operations necessary to maintain the integrity of the data model by controlling the side effects are automatically performed in the CODASYL type DBMS.

The purpose of the mapping rules is to map the E-R conceptual model on to the logical structure of a relational or CODASYL type DBMS. It is independent of any particular implementation. Before we can implement the data model on to a definite DBMS, the physical characteristics of the data have to be considered.

CHAPTER 7

PHYSICAL DESIGN

7.1 Introduction

The objective of the mapping of the E-R conceptual model on to a relational or CODASYL logical model is to capture all the information and to free the logical model from all update anomalies. The objective of the physical design is to develop an efficient and implementable data base structure which will satisfy the user requirements.

The performance of a DBMS is related to the speed with which the DBMS responds to requests and this is directly dependent on how efficiently the data is stored. The performance can be modified by changing the physical structure of the data base to suit the users' needs. Most DBMS offer the data base administrator (DBA) options to structure the data physically so as to enhance the performance and support a number of access methods. The DBA should be able to specify how to store the data on the storage media, the sizes of the memory buffer, etc. However, when these options are not available the DBA has to make the best of what is provided by the particular DBMS.

The physical design phase looks at how the appropriate logical data model can be physically organised so that it will have an impact on the performance of the implemented data base. This phase involves structuring the stored data and the result obtained is a data base structure ready for implementation. There are other options available, which can be used to improve the efficiency of an implemented data base. This project focuses on the organisation of the data base so as to get a good performance from the implemented data base. In this chapter we will discuss the features that affect file organisation, different ways of organising data and algorithms for assigning a suitable data organisation.

7.2 Factors affecting file organisation

The most important criterion that decides how we organise a file, is the way in which the file is referenced by the user. The cost of processing a file also plays an important role. Factors that affect the cost are: the frequency of file reorganisation, the storage space required, the programming costs, etc. Other factors which should influence the choice of file organisation are:-

Processing facility: that is, whether the file is being processed in batch or for on-line queries or both

Hit rate:	that is, the percentage of records being processed in one run.
Size of the file:	that is, the number of records and the length of each record.
Growth:	that is, how often and how many records are being added or deleted from the file.

The average time required for an amendment, an update or to process a query also affect the design decisions.

7.3 Organisation of data

The performance of the data base depends on how accurately the conceptual model is mapped on to the logical model of the target DBMS, and also on how well we utilise the physical characteristics of the DBMS.

The details noted in the functional analysis phase are used to decide the file organisation methods and ways of accessing records within the files. Factors such as how frequently an access path is used, what volume of data is retrieved, what response time is necessary and for what category of function (primary or secondary) a particular access path is being used, affect the physical organisation of the data.

At this stage, we have to decide from our functional data model which access paths should be supported. The characteristics of the access paths decide how data is to be placed and related so as to give fast access. Let $X \to Y$ be an edge, which may represent a relationship edge or a property edge. There are three possible methods of organising the elements of X and Y to improve access time.

1. Indexed:- Given y in Y, $f^{-1}(y)$ can be found without scanning all the elements of X.

The relationship 'TREATED BY' will be indexed if there exists a fast access path between 'CONSULTANT' and the occurrences of 'PATIENT' treated by each 'CONSULTANT'. The entity 'PATIENT' may be indexed on the attribute 'Pat-no' if it is inverted on that field.

2. Clustered:- For every y in Y, all the elements $f^{-1}(y)$ in X, will be close together, so that, accessing one element of $f^{-1}(y)$ will ensure a quick access to the rest of the elements in $f^{-1}(y)$.

Relationship 'TREATED BY' is clustered if all the occurrences of 'PATIENT' treated by a particular 'CONSULTANT' are kept contiguously. The entity 'CONSULTANT' may be clustered on the attribute 'speciality', if all the occurrences of 'CONSULTANT' with the same speciality are kept together.

3. Well-placed:- It is a similar concept to clustered, the only difference being that all the elements of $f^{-1}(y)$ are kept close to y.

Relationship 'TREATED BY' will be well-placed if all the occurrences of 'PATIENT' 'TREATED BY' the same 'CONSULTANT' are kept together near the associated consultant. The entity 'CONSULTANT' will be well-placed on 'speciality' if it is clustered on the attribute 'speciality' with the speciality value taken out of the records.

7.3.1 Priorities of data organisation

Among the data organisation methods discussed, the well-placed gives the fastest performance, followed by the clustered and then the indexed structure. Ideally, it would be preferred that for all the edges, the $f^{-1}(y)$s are well-placed near the corresponding y. This is not feasible, because there are a few constraints which apply. Taking these into account, it is preferable to have a well-placed structure where possible, if not then a clustered, and, if this were not feasible, an indexed structure. From this a priority to these structures can be assigned as following:

1) Well-placed 2) Clustered 3) Indexed.

7.3.2 Constraints on data organisation

The objective at this stage is to study the functional data model and investigate the paths those need to be supported. Then it is necessary to assign maximum possible support for each of these paths. Assignments of type of support will of course be subject to the constraints, which will be discussed in the following paragraphs.

i. First constraint (Figure 7.1)

Let X,Y and Z be three nodes from our data model. If the elements from Y are clustered and then placed near their associated element in X, it will not be possible to do the same with the elements of Z. The elements of Y will prevent the Z elements being placed in close proximity to X. Hence, it will not be possible to assign well-placed structure to more than one inedge of a node.

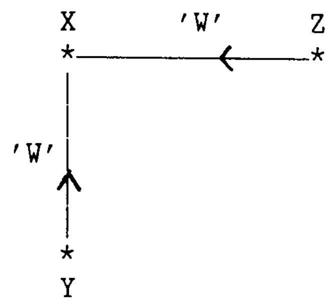


Figure 7.1 First Constraint

ii. Second constraint(Figure 7.2)

In the above example, if the elements of X which share the common element of Y are clustered, it will not be possible to cluster the elements of X which share the common element of Z. Thus it is not possible to assign more than one outedge in a clustered structure.

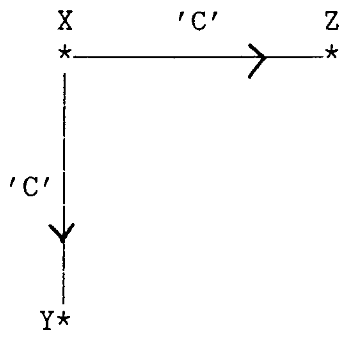


Figure 7.2 Second Constraint

iii. Third constraint(Figure 7.3)

The clustering of Y elements near the corresponding X elements prevent clustering elements of X which share the same Z together. Thus, it is not possible to assign, simultaneously, an inedge of a node well-placed structure and the outedge a clustered structure.

$$\begin{tikzcd}
X \arrow[r, "'C'"] & Z \\
Y \arrow[u, "'W'"] &
\end{tikzcd}$$

Figure 7.3 Third Constraint

7.4 Labelling the functional data model

To be able to define the physical data base structure it is necessary to allocate these data organisations to our data model. We have to define algorithms for assigning labels to the edges in the data model so as to achieve the best possible access performance. The labelling should be such that the data model is free of the constraints mentioned in the section 7.3.2.

The labelling method used in this project was as following:

i. Label each edge as indexed.

ii. Traverse the model again according to the frequency order, and label the edges as clustered as long as the model is free of the constraints mentioned in section 7.3.2.

iii. Traverse the model again in frequency order and label the edges as well-placed, ensuring that the model is free of the specified constraints.

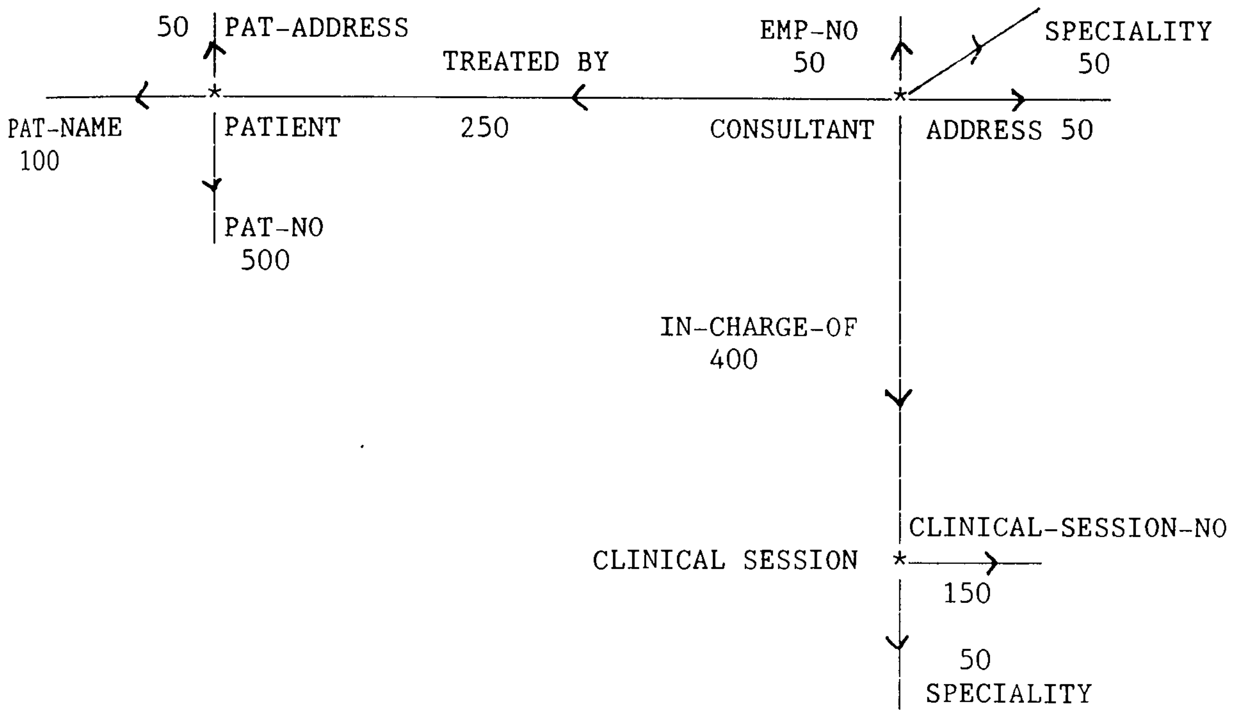


Figure 7.4 Unlabelled Data Model

Applying the algorithm to the data model in Figure 7.4, the labelled model Figure 7.5 is obtained.

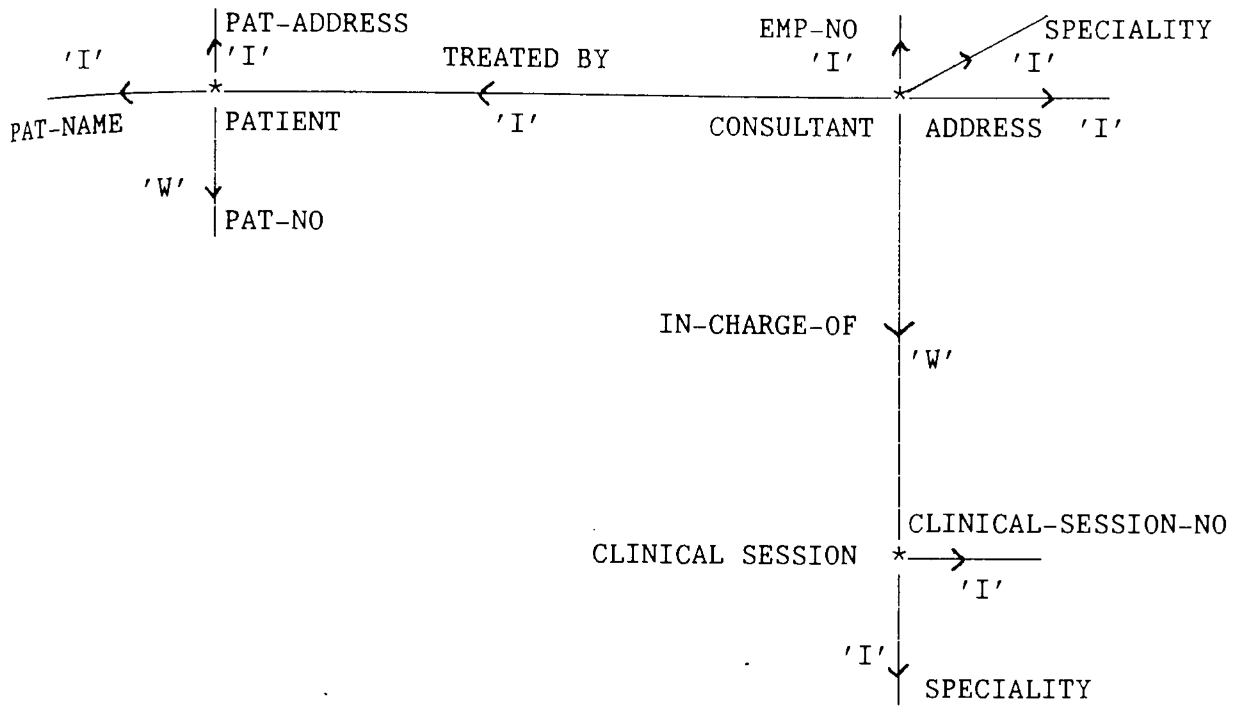


Figure 7.5 Labelled Data Model

The algorithm ensures that, wherever feasible, the edges have been labelled with the highest priority label. It is not possible to prove mathematically whether the optimal solution is achieved by this method. This is a problem with most of the applications of directed graphs. This algorithm does not directly take into account for which category of function a particular access path is being used and what sort of response time is necessary for this function. This algorithm assigns best possible support to the paths which are used most frequently.

Program, 'LABEL.P', follows the above algorithm and labels the edges with the appropriate labels. It examines the result of the functional analysis and the lists of entities and relationships, and then assigns the necessary data organisation labels.

The program, 'LABEL.P', creates two files, 'edgefile' and 'labelfile'. The 'edgefile' contains the list of all edges and describes each edge in terms of the frequency of use, its type (that is, whether it is an attribute, a primary key or a relationship) and the label assigned to this edge, following the labelling algorithm. The 'labelfile' describes each entity in terms of all the inedges coming into the entity node and outedges going out from the entity node.

The program first reads in the entities from the 'entfile', the relations from the 'relfile', the function details from the 'funcfile' and the usage details from 'eusagedet' and 'rusagedet'. Once the relevant files have been read in, the components of 'labelmat' and 'edgemat' are determined.

The variable, 'labelmat', describes each entity node in terms of the number of outedges and inedges, the usage frequency of each of the edges and the labels assigned to each of these edges. It is represented by an array of the record 'graphdet'. The record, 'graphdet', has the following components:

noofoutedge	an integer denoting the number of outedges the entity node has
noofinedge	an integer denoting the number of inedges the entity node has

outedge	a description of each outedge, represented by an array of the record 'outdet'
inedge	a description of each inedge, represented by an array of the record 'indet'

The record, 'outdet', has the following components:

oedegetype	a character denoting whether the edge is a primary key, an attribute or a relationship
oedegepnt	a pointer to the appropriate attribute or relationship
oedgefreq	an integer denoting the usage frequency of the edge
oedgelabel	the data organisation label assigned to the edge

The record, 'indet', has the following components:

iedgepnt	a pointer to the relationship
iedgefreq	the usage frequency of the inedge
iedgelabel	the data organisation label assigned to the edge

The value of the components of 'labelmat' are obtained from the files that are read in initially. However, the labels can only be assigned by applying the labelling algorithm described in section 7.4. The initial value assigned to the labels is 'I' for Indexed. The value of the components of 'edgemat' are also filled in this phase.

The matrix, 'edgemat', describes each edge in terms of its label, usage frequency and edge type. It is represented as an array of the record 'edgedet' which has seven components. The first three describe the label of the edge, frequency of the edge and the type of the edge respectively. The last four are pointers which are used to identify the edge.

To be able to assign the data organisation labels, the records of 'edgemat' are sorted according to frequency and stored in 'sortlist'. Then each record in 'sortlist' is examined and if possible is assigned a label 'C'. The list is examined again in frequency order and wherever possible a label 'W' is assigned. Once the labelling procedure is complete, 'edgemat' is stored in the 'edgefile' and 'labelmat' is stored in 'labelfile'.

7.5 Physical design rules

So far, it has not been specified how the clustering is achieved. There are two possible solutions. We can either have an indexed sequential structure or apply some hashing technique. The choice depends on the amount of data retrieved in one particular access. If a subset of a file is retrieved in one particular access then an indexed sequential structure should be chosen. If only a single record is retrieved, a hashing technique should be applied. The information on the volume of data retrieved is obtained as a result of functional analysis. In cases where a particular access path or edge is used to retrieve a single record or a subset of records, an indexed sequential structure can be chosen. This will support both types of retrieval successfully. Paths which are used frequently, but cannot be supported by the methods described so far, are supported by secondary indices. It is also assumed that if a path is being traversed frequently it is being used for primary or essential applications which use the data base.

The results obtained by applying the algorithm are consistent with many users' experiences. Severance and Carlis (1977) mentioned that one of the methods to improve the performance of a data base system is to increase the volume of data transferred with each data access. This could be achieved in more than one way. One of the techniques could be to store contiguously, records which are processed sequentially. They have described how records could be physically positioned according to the speed of response required and the volume of data retrieved. Their views are summarised in Figure 7.6.

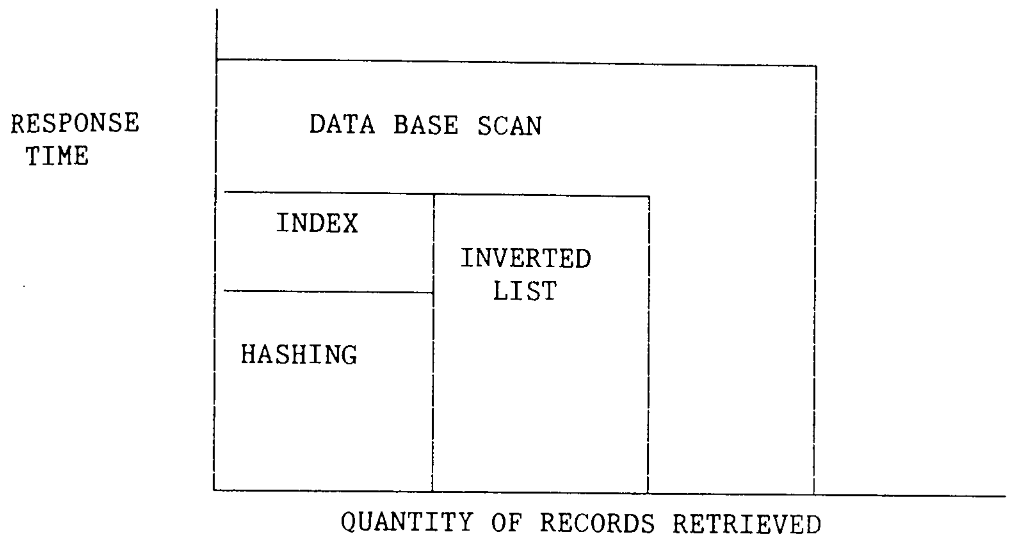


Figure 7.6 Record Accessing Techniques

There are three approaches to retrieving an individual record. If a fast response is essential, a hashing technique should be used. Rather slower than this would be the use of an indexed sequential structure and access via indices. Thirdly, if a speedy response is not deemed to be important, a data base scan will be satisfactory.

If we are accessing several records (ie, a subset of the entire file) and a fast response is needed, an ISAM structure may be chosen. Where time is less critical, a data base scan can be performed.

7.6 Conclusion

There are many ways of organising data to improve performance. Factors affecting data base design include, among other things, the frequency of the reorganisation of a data base, the amount of free space, and the allocation of indices to another fast device. Once a particular DBMS has been chosen, some of the options available to improve the performance become restricted. However, the DBA can further improve performance by organising data in a satisfactory way. We have focused in this chapter on the organisation of the data base files as a function of usage.

CHAPTER 8

RELATIONAL IMPLEMENTATION

8.1 Introduction

Many DBMS describe themselves as relational systems. Their claim may only be valid in that that they present the data in the form of a table. Kim (1979) has proposed a number of valuable features for relational systems, many of which are general features of any true DBMS. The requisite features of relational systems as defined by Kim (1979, pp 185-186) are:

' 1. An interface for a high level, non-procedural data language that provides the following capabilities for both programmers and non-technical users: query, data manipulation, data definition, data control facilities.
2. Efficient file structures in which to store the data base and efficient access paths to the stored data base.
3. An efficient optimizer to help meet the response-time requirements of terminal users.
4. User views and snapshots of the stored data base.
5. Integrity control-validation of semantic constraints on the data base during data manipulation and rejection of offending data manipulation statements.
6. Concurrency control-synchronization of simultaneous updates to a shared data base by multiple users.
7. Selective access control-authorization of access privileges of one user's data base to other users.
8. Recovery from both soft and hard crashes.

9. A report generator for a highly stylized display of results of interactions against the data base and such application-oriented computational facilities as statistical analysis.'

Date (1986) classifies relational DBMS into the following four categories depending on the facilities supported by them:

Tabular: A tabular DBMS presents the data to the users in a tabular form but does not support any of the relational algebra operations like SELECT, JOIN, PROJECT, etc.

Minimally relational: A minimally relational DBMS supports the tabular structure and some of the relational algebra operations but not the full set of relational operators.

Relationally complete: A relationally complete DBMS supports the tabular structure and the full set of relational operators.

Fully relational: A fully relational DBMS supports the tabular structure, complete set of relational operators and the two integrity rules, namely entity and referential integrity.

By entity integrity it is meant that no attribute which participates in the primary key may have a null value. Referential integrity means that if a relation has a foreign key then its value must match the primary key value of one of the rows in another relation or must have a null value.

Currently, there are no DBMS available that are fully relational. According to this definition, DB2 from IBM, ORACLE from ORACLE Corporation, MIMER from Savant, INGRES from Relational Technology, etc, fall into the category of relationally complete DBMS. The two DBMS used in this research are INGRES and MIMER. INGRES and MIMER were chosen because they support physical file structures which are different from the others and will therefore highlight the issues associated with different physical designs. These are described in section 8.2.2 and 8.3.1. Moreover, INGRES is portable across many operating systems and is widely used in academic and commercial organisations.

8.2 INGRES

INGRES (Relational Technology Inc, 1983) runs on VAX Data General and Hewlett Packard 9000 series machines, IBM mainframes and IBM PCs, NCR Tower, etc. Before describing the mapping and design rules on this particular implementation, a brief description of the system will be given.

8.2.1 Basic structure

A data base implemented on INGRES contains a number of relations. A relation possesses a number of attributes. A relation in INGRES cannot have more than forty-nine attributes.

When a relation is created the format of each attribute should be specified. For example,

```
Create Patient
            (Pat-num =  i8
            Pat-name = c20
            Pat-add =  c40)
```

will create a new relation Patient with three attributes: Pat-num, Pat-name and Pat-add. Pat-num is an eight byte-integer, and Pat-name and Pat-add are 20 and 40-bytes character. Once a relation is created and data is inserted, it can be used in response to any relevant query.

8.2.2 Storage structures

INGRES supports five storage structures. Four of them are keyed, i.e, the location of the tuple is dependent on the value of the primary key. They are termed 'Hashed', 'ISAM', 'Compressed Hash' and 'Compressed ISAM'. The nonkeyed structure is termed 'Heap' and the tuples are stored independently of primary key values. Definition of the storage structures are given below.

1. Heap

There are two main characteristics of a heap structure.

i. Duplicate tuples are not removed.

ii. The organisation of the tuples is not known.

To retrieve a particular tuple, the system has to scan every tuple until it finds one tuple which satisfies the condition of the search. This type of search is not efficient if it is being done on a vast amount of data.

2. Hash

In this structure all duplicate tuples are removed and the relations are hashed on a specific domain. The system will provide fast access to that tuple if the value of the domain is provided, because the location of the tuple on the disk will be known to the system.

3. Indexed Sequential Access Method (ISAM)

The duplicate tuples are removed in this structure. The relations are sorted on one or more domains. Any retrieval will be enhanced if the value(s) of the domain(s), or the range(s) of the value within which the domain(s) lies(lie), is provided.

Both 'hash' and 'ISAM' structure can be operated on a 'compression' mode. This mode suppresses the blank and portions of tuples which match the previous tuples. These structures take longer to update, but economise on storage space.

The designer creating the relations is able to specify what storage structures are required. In the event of not specifying the storage structure, the system assigns the default structure of heap.

Both hash and ISAM assign specific tuples to a particular page on disk. When inserting new tuples, if the system finds that the primary page is filled, it stores the tuple in the overflow area. The creator can specify how full to make a primary page initially by indicating the fill factor. When hashing a relation, the user can also specify the minimum or maximum number of primary pages to allocate to the relation. The person creating the relations should have prior knowledge of how the relations are going to grow.

A program called 'Sysmod' is incorporated in the INGRES system and this should be run initially when a data base is created. As the data base grows, this program is run at regular intervals to improve efficiency.

8.2.3 INGRES implementation

8.2.3.1 Implementation of the logical model

From the discussion in the previous section, it can be stated that the relations obtained by applying the mapping rule can be mapped straightaway on to the relations in INGRES. All the relations are in the fourth normal form and have no replication. Therefore, there will not be any problem in performing this mapping. Initially, all the relations will have a heap structure. No major problems will be encountered in using this data base. Nevertheless, this will not run as efficiently as desired. To improve the performance it will be necessary to modify the storage structure of the relations. Modification decisions rest on the results obtained in the physical design phase.

8.2.3.2 Implementation of the storage structure

INGRES does not support physical access paths between relations. Instead, the primary key of an entity is added on as an attribute or foreign key of another entity. For example (Figure 5.2), there is no physical access path from the relation 'COAG-REFERRAL' to the relation that represents the entity 'PATIENT'. Instead, the primary key of the 'PATIENT' entity is added on as an attribute of the relation 'COAG-REFERRAL'. This denotes which report belongs to which individual patient. Thus, in the physical design model, the frequency of the identifier edge of an entity must also include the frequencies of all the 'n:1' and '1:1' relationship edges, with this entity in the range. These frequencies are represented by the frequencies of all the inedges of the entity. These adjustments should be performed before a decision on the primary structure of the relations can be made.

The other relevant limitation of INGRES is that tuples from different relations cannot be placed near one another. Hence it is not possible to cluster the entity occurrences which are associated with another common entity occurrence, around the entity of the latter entity occurrence. For this reason, it is not possible to label any outedge of an entity 'W'. As pointed out in section 7.3.2, one outedge at most can be labelled 'W' or 'C'. In this case the outedge will have to be labelled 'C'. As specified in section 7.3.1, because 'C' has a higher priority than 'I', this edge will normally dictate the primary structure of the relation.

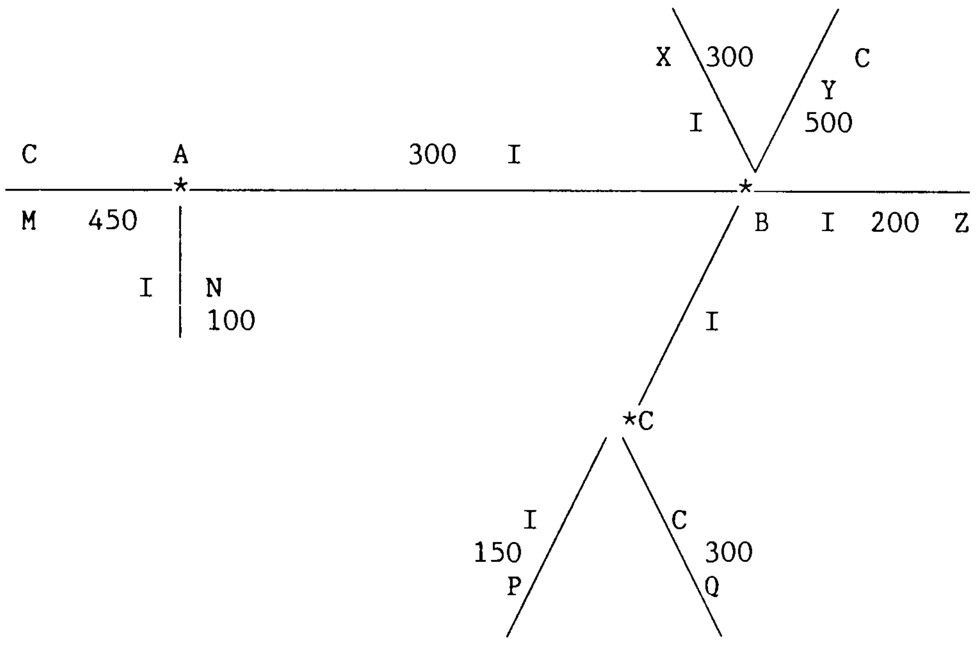


Figure 8.1 Labelled Model

In the example shown in Figure 8.1, the edge Y has the label 'C'. The relation will be clustered on this attribute. This might not always be true. As specified earlier (section 8.2.3.2), the frequency of the identifier edge should also include the frequencies of all the inedges coming into the entity. If the summation of these frequencies and the frequency of the identifier outedge is greater than the frequency of the outedge labelled 'C', then the identifier or the primary key forms the basis of the primary structure of the relations. The chosen edge represents either an attribute or a relationship. If the chosen edge is an attribute then the storage is structured on the basis of this attribute. If the edge represents a relationship 'X→Y' of degree 1:1 or n:1, and the membership of X is obligatory then the identifier of Y forms the basis for the primary storage structure of the relation R(X). If the edge represents a relationship 'X→Y' of degree n:1 where the membership of X is not obligatory, or a relationship of degree m:n, then there will be another relation in the logical model which represents this relationship. The primary storage structure of R(X) will then be based on the identifier of X. The traversal frequency of the edge representing the relationship 'X→Y' will now have to be added on to the frequency of the identifier edge. The primary structure of the relation R(XY) will be based on the identifier of 'X' or 'Y', depending on which of the frequencies between 'X→Y' or 'Y→X' is greater. The other identifier can be used for an optional index to the relation.

In the previous section, we mentioned only the field on which to base the primary structure of the relation. Whether we choose an ISAM structure, which supports a range access, or a hash structure which supports only equality access, is determined from the information on how the access paths are used. Identifier edges are often used in equijoin clauses, that is, the particular field is matched with the appropriate field in a query. The hash structure is suitable in this situation. A property edge is most often used in range access, therefore an ISAM structure is appropriate. Of course, there might be cases where the property edge is used frequently for equality access. In these circumstances, a hash structure is suitable. When in doubt, or the particular edge supports both equality and range access, then the ISAM structure is appropriate. Edges labelled 'I' are supported by secondary indices. The designer should be able to decide from the frequencies whether an edge labelled 'I' needs to be supported or not.

8.2.3.3 Algorithm for the storage structure

From the above discussion we can formulate the following algorithm.

```
Let x = identifier outedge
    y = other outedge labelled 'W' or 'C'
    z1.....zn = inedges
    fn = frequency of n
```

```
IF the relation represents an entity

THEN

        IF fx + fz > fy
           THEN hash on identifier attribute
        ELSE IF y is a property edge
             THEN ISAM on property value
        ELSE IF y is a relationship edge represented
                 by a foreign key
             THEN hash on foreign key
        ELSE hash on identifier edge;
   FOR each outedge labelled 'I' create secondary index;

ELSE

IF the relation represents a relationship

  THEN

       Hash it on the identifier of the entitya
       Have secondary index on the identifier of the
            entityb.
```

The above algorithm is a logical process. A program, 'ING.P', will take as input the relations obtained by applying the mapping rule, the field types from the entity list, and the frequencies and labels of the access paths, and produce relations with the necessary storage structure, ready for implementation on INGRES.

The program, 'ING.P', creates a list 'ingentchart', which is represented by an array of the record type 'ingent'. The record type, 'ingent', has three components, they are:-

struc	a character representing the storage structure
edgeno	an integer pointing to the appropriate edge
diredge	a character representing the direction of the edge.

By examining 'norentchart' and 'labelmat', 'ingentchart' is created. There is an entry in 'ingentchart' corresponding to every normalised entity. If an outedge is labelled 'W' or 'C' and the sum of the usage frequencies of the inedges is greater than the frequency of the outedge marked 'W' or 'C', then the structure is labelled 'H', 'edgeno' points to the primary key and the direction of the edge ('diredge') is 'o' representing an outedge.

If the outedge is labelled 'W' or 'C' and the sum of the usage frequencies of the inedges is less than the frequency of the outedge marked 'W' or 'C' and the outedge represents an attribute then the structure is labelled 'M', 'edgeno' points to this specific attribute and 'diredge' is 'o'. On the other hand, if the outedge marked 'W' or 'C' is the identifier then 'struc' becomes

'H', 'edgeno' points to the primary key and 'diredge' is 'o'.

If an inedge is marked 'W' or 'C', then 'struc' becomes 'H', 'diredge' points to the foreign key of the entity and 'diredge' is assigned the value 'i' so that it represents an inedge (a relationship edge).

Once all the entities have been examined, they are written on to the 'mapfile' which is the structure ready to be implemented on INGRES. The normalised entities with their attributes are written on the file. Moreover, 'ingentchart' is examined and the file structure is recorded from there. If 'struc' is 'H' the relation is hashed on the primary key or the foreign key depending on 'edgeno' and 'diredge'. If 'struc' is 'M' the relation is inverted on the attribute pointed by 'edgeno'.

Each normalised relation representing the relationships are also written on to the 'mapfile'. These relations are all hashed on the primary key of 'entitya'.

All the programs for this research are written in an interactive form. In situations where the decisions are doubtful, the designer is given the chance to make a decision. The designer can be aware of this and in due course of time, can try out the other options available. This will enable him to conclude which options give a better performance.

INGRES does not have provisions to deal with replicated data. Replication, to a certain extent, improves efficiency but produces problems during updating. Special programs have to be written to deal with these side effects. In this project replication of data has not been dealt with.

So far, the choice of storage structures has been discussed. Hash is advantageous for locating tuples where the exact value is known. ISAM is useful for both range and exact values. In an ISAM structure the directory has to be searched before we can locate the required tuple. Therefore, it is not as efficient as hash for locating tuples when the exact value is known. When a tuple is inserted, the system first finds the page where the tuple should belong. If the page is not filled then it stores the tuple in that page. Otherwise, it stores it in an overflow page. Too many tuples stored in an overflow area affect the performance of the data base. Initially, when we are creating a relation and appending the tuples, it is easier to keep the relations in a heaped structure. Once this is completed, we can modify the relations to the structure necessary. We can specify the 'fillfactor' in the case of ISAM and hash structured relations.

The INGRES Reference Manual (Relational Technology Inc. 1983) describes how, if we have knowledge of the maximum number of tuples in a relation and the existing number of tuples, we can specify the 'fillfactor'. Let 'y' be the maximum number and 'x' be the existing number of tuples, then we can assign a 'fillfactor' of (x/y)*100. This is a very rough guide and it assumes that the tuples are evenly distributed.

If we know the number of bytes in each page, the overhead of bytes per page and per tuple, the tuple width and the maximum number of tuples, we can specify the minimum number of pages required for a hash structure.

Let the number of bytes in a page = x

Let the number of bytes for overhead per each page = y

Let the tuple width = w

Let the tuple overhead = z

Maximum number of tuples = n

Number of tuples in a page(t) = $(x-y)/(w+z)$

Minpage = n/t

This will guarantee that free space is reserved for the relation to grow to its maximum size. When the 'fillfactor' and 'minpage' are both specified, the system computes the number of pages that will be required to store the existing tuples at the specified 'fillfactor'. If this number is less than 'minpages', then 'minpages' is used instead.

8.3 MIMER

MIMER (Savant Enterprise, 1985) developed in Uppsala University Data Centre is a relational DBMS which is efficient for handling routine sequential processing and random information retrieval. It has an integral data dictionary which controls data access, usage and security.

8.3.1 MIMER storage units

A MIMER DBMS consists of a number of 'databanks' (data bases). Each databank consists of relatively addressed pages, whose size is dependent on the particular installation. Each databank contains a number of 'n-ary' relations. A databank may contain the contents of more than one table, but the contents of a table are always stored in one databank. The first page of a databank is the bit map. This indicates which pages are free and which pages are used. The second page is the root page, which keeps a directory of all the tables in the databank. If the databank is large, then more than one root page or bit map may be necessary. Data is transferred from secondary to primary storage on a page to page basis. The relations are stored on the disk on a B*-tree structure, sorted according to the values of the primary key.

8.3.1.1 B-tree structure

Teorey and Fry (1982, pp 306) define a B-tree as

' ... a generalization of a binary tree in which two or more branches may be taken from each node.'

B-tree structures are efficient for storing large indices and for random access of records. A B-tree structure has the following features:-

1. All the paths from the root to the leaf nodes are equal and that is termed the 'height' of the tree.

2. All the nodes (except root and leaves) of a B-tree of order n have at least n+1 child nodes and maximum 2n+1 nodes.

3. The root node has at least two child nodes and a maximum of 2n+1 child nodes.

4. Each node (except the root node) can have at least n keys and not more than 2n keys.

B-trees are more flexible than binary trees because of the variable number of keys allowed in each node. The height of a B-tree tends to be smaller than a binary tree and thus it allows faster retrieval time. B-tree offers good performance for random access as well as sequential accessing, without having to reorganise the file.

8.3.1.2 B*-tree

This is a variation of a B-tree, and has all the keys and the associated data in the leaf nodes and the records are accessed via an index which is a B-tree index. The leaf nodes of a B*-tree are formed of keys, associated data and a pointer to the next level node. In the initial loading, all keys are stored in the leaf nodes and the keys in the index are duplicates of these keys.

As far as performance is concerned there is not much difference between the B-tree and B*-tree organisations. However, B*-trees are more efficient for the sequential accessing of large volumes of data. Detailed descriptions of these structures can be found in Teorey and Fry (1982) and Comer (1979).

8.3.2 Physical storage structure of MIMER

The B*-tree file structure in MIMER consists of two sections:

a) The data sections - which contain the leaf nodes of the tree and are the rows of the n-ary relations.

b) The index sections - which contain the non-leaf nodes of the tree and are the navigational paths to the nodes in the next level.

This provides a fast binary search technique to access a particular row of a table or a place where the row should be inserted. As the table grows, the MIMER DBMS automatically reorganises the tree structure and a periodic reorganisation is not necessary. The tables of data are sorted and stored according to the primary key value. It is quicker to locate a row of the table if the primary key is known. Thus, the structure is suitable for random access. This structure can also process records sequentially unlike the hash structure.

It is also possible to have one or more secondary indices to a table which invert the table on one or more columns. When a secondary index is created the MIMER DBMS forms an internal table with the specified column sorted and registers the corresponding primary key with it. When a table has secondary indices it is faster to retrieve data but slower to update.

The columns or the fields in the MIMER relation can be of three types: characters, integers or floating point numbers. Each data type can be of any specified length.

8.3.3 Implementation on MIMER

8.3.3.1 Implementation of the logical model

From the above discussion, we can see that the MIMER DBMS contains a number of n-ary relations. The relations obtained by applying the mapping rules can be directly mapped on to the relations in the MIMER DBMS. The process will be similar to the INGRES mapping. The rows for the relations are held in a B*-tree structure, based on the primary key of the relations.

A fast binary search technique will retrieve a row, if the primary key is known. Secondary indices can be added so that it is also possible to retrieve a row when any field other than the primary key is known.

8.3.3.2 Implementation of storage structures

MIMER, like INGRES, does not support the physical access paths between relations. For the reasons discussed previously (section 8.2.3.2), the frequency of the identifier edge in the global functional model should also include the frequencies of all the inedges coming into the entity node. For similar reasons to those applicable to INGRES, it is not possible to have edges labelled 'W'. Therefore, at most, one outedge will be labelled 'C' and this will dictate the primary storage structure of the relation.

In the case of MIMER, the DBA has very little control on how to organise the storage structure. The rows are stored automatically as a B*-tree based on the primary key. The only other option available to the DBA is to add secondary indices based on attributes through which the search is to be conducted. When a secondary index is created, an internal index table is set up. The

8.3.3.3 Algorithm for the storage structure

From the discussion so far, we can formulate the following algorithm.

```
Let x = identifier outedge of a node
    y = other outedge labelled 'W' or 'C'

FOR every node DO
    { IF y is a property edge
          THEN invert R(X) on this attribute }

      ELSE
          { IF y is a relationship edge XY of
                              degree 1:1 or n:1
            and membership of X is obligatory
            THEN invert R(X) on the identifier of Y }

            ELSE invert R(XY) on the identifier of Y
```

Program 'MIM.P', like 'ING.P', produces a data structure suitable for MIMER implementation. The program, 'MIM.P', creates a list 'mimentchart'. The list is represented by an array of the record type 'miment'. The record type, 'miment', has three components, which are:-

struc	a character representing the storage structure
edgeno	an integer pointing to the appropriate edge
diredge	a character representing the direction of the edge.

By examining 'norentchart' and 'labelmat', 'mimentchart' is created. There is an entry in 'mimentchart' corresponding to every normalised entity. Whenever a record is stored in the MIMER DBMS, it is sorted according to the value of the primary key. The only structure that the designer can enforce is to create secondary indices on some attribute.

The program, 'MIM.P', examines every entity node. If an outedge which represents a nonkey attribute is labelled 'W' or 'C', then 'struc' becomes 'I', 'edgeno' points to the attribute and 'diredge' shows that it represents an outedge.

If an inedge is labelled 'W' or 'C', then 'struc' is labelled 'I', 'edgeno' points to the 'entitya' of the relationship that is represented by that edge and 'diredge' shows that it represents an inedge.

If none of the above conditions holds, then the edge labelled 'W' or 'C' will represent the primary key and there is no need to impose any additional structure on this relation.

The program then copies all the relations representing the normalised entities on to the 'mapfile' ready for MIMER implementation. Each entity, along with its attributes, is copied to this file. Then 'mimentchart' is examined. If 'struc' for this entity is 'I' then that relation is inverted on the attribute indicated by 'edgeno'.

The relations representing the relationships are then copied on to 'mapfile'. They are all inverted on the primary keys of the 'entitya' of those relationships.

8.4 Conclusion

At the end of this phase, basic guidelines for implementing the relational model on to a particular relational implementation like INGRES and MIMER are obtained. From the storage algorithm it can be seen that the DBA has significantly less control on the storage structure of a data base implemented on MIMER than a data base implemented on INGRES, due to the lack of storage options available in MIMER DBMS.

The B*-tree structure of MIMER expands and shrinks, automatically reorganising itself, as records are added to, or deleted from, the data base. Therefore, it does not need regular reorganisation. In contrast, the performance for INGRES will deteriorate with the addition and deletion of tuples, and the DBA will need to reorganise the data base from time to time.

CHAPTER 9

CODASYL IMPLEMENTATION

9.1 Introduction

The VAX-11 DBMS (see Digital Equipment Corporation 1981a, 1981b) is a CODASYL type DBMS based on the 1981 working document of the ANSI Data Definition Language Committee (ANSI X3H2 1981). VAX-11 DBMS follows the principles established for a generalised DBMS. It provides a centralised location for the data definitions and the physical characteristics of the data are separated from the logical data definitions.

The VAX-11 DBMS uses data definition languages (DDLs) to define data and data base constructs. It has three DDLs, namely, a schema DDL, a subschema DDL and the storage schema DDL. Future implementations of VAX-DBMS are going to include a further DDL called the security schema DDL. A <u>schema DDL</u> is the overall logical definition of a data base. A <u>subschema DDL</u> describes a subset or user views of the data base. A <u>storage schema DDL</u> describes the physical storage structure of the data base. A <u>security schema DDL</u> describes the access constraints on particular data. This research looked at automatic generation of a schema and storage schema definition. These definitions will be discussed in more details.

The logical data definitions are centralised in the schema. As discussed previously, in section 6.2, the logical structure of the data is defined in terms of records and sets. In earlier implementations of CODASYL, the schema contained definitions of the physical and logical characteristics of the data. In this particular implementation, the schema contains only the logical characteristics, and the physical characteristics of the data are defined in the storage schema. The schema contains the description of the records, the definition of sets, the owner and member records for each set, and the order and the insertion/retention clause of the member records. The storage schema contains the storage characteristics of the records and sets. It describes how records should be placed in the data base.

9.2 Schema definition

The schema describes the logical characteristics of the data. The schema definition includes the description of areas, records, sets and ordering of member records within a set. These concepts are similar to the ones put forward by the CODASYL committee. In the following section a brief description of these concepts, as described in this particular CODASYL implementation, is given.

Areas	Schema areas are logical subdivisions of the data base. The areas are small manageable units which isolate applications to restricted areas.

Records	The schema record description includes the name of the record, its component data items, their types and the area within which it lies. A record may occur in more than one area.
Sets	The schema set definition specifies the relationship between records. It defines the owners and members of sets. The insertion and retention clause of member records are defined. The options available for the insertion and retention clause are the same as the ones described in section 6.2.2.
Ordering new members	This defines where a new member record is inserted. All member records are connected to the owner record through pointers. When a new member is stored, depending on the ordering, it is inserted in the appropriate position in this chain.

9.3 Storage schema

The storage schema contains information about how the logical records defined in the schema are physically stored in the files. If the storage schema is not specifically defined, the DBMS assigns a default storage schema. The performance of the data base can be improved if the storage options available are utilised to suit the processing needs. The facilities available for storing records and sets in the VAX-11 DBMS will be discussed in the following section.

9.3.1 Storage records

There are two basic record placement options available in VAX-11 DBMS. They are to cluster records around the owner a set or to scatter records throughout the data base.

Clustered via records	Records may be clustered around the owner record of a set of which it is a member. That is, records which are associated with the same owner record may be placed near the owner record. The insertion class of this member record in the set has to be defined as automatic. Clustering is beneficial if the number of the member records is not very large. Clustering is useful when our processing needs are such that we access the records through the owner record.
Scattered records	In this option the records are scattered uniformly throughout the storage areas depending on the key of the records. The owner record of a set whose members are clustered around it, is best distributed uniformly throughout the storage area.

9.3.2 Storage set

The storage sets describe how the DBMS stores the member records. The three options available are chain, calc and index.

Chain sets	This mode is useful for small unsorted sets where it is necessary to process all members serially. It is not efficient for a quick direct search for a member record via the owner record of a set.
Calc sets	This storage mode can only be defined for members of a system-owned set. The system uses the key or some data item of the member record to find the location of the record.
Index sets	This mode can be used for sets whose members have been sorted. The records are sorted according to a particular data item and the retrieval is efficient if this data item is provided by the user.

9.4 VAX-11 DBMS implementation

Like any data base implementation, implementing a data base on the VAX-11 DBMS involves mapping the logical model on to the VAX-11 DBMS schema and then defining a storage schema based on the results of our functional analysis.

9.4.1 Schema mapping

The record and set structure obtained by applying the mapping rules described in section 6.4, can be mapped on to the schema definition of this implementation. For every record in the CODASYL data model, there will be a corresponding record defined in this schema definition. For every set defined in the CODASYL data model, there will be a corresponding set defined in this schema. The insertion and retention clauses are also available from the logical model.

The concept of system-owned set structure was not discussed in the logical model defined in section 6.2. This does not strictly represent the semantics of the data. The concept of the system-owned set exists to facilitate access capability. Record types with large numbers of members, which act as entry points to the data base, are defined as members of system-owned sets. This provides quick direct access to the records through the key data items. The information needed to take this decision is not available from the E-R data model. Therefore, the record-set structure obtained by applying the mapping rules, does not specify the sets owned by the system. The results obtained from our functional analysis specify the number of records in a record type and the access pattern of the record type. This information can be used to define the system owned set when implementing the record set structure on this particular implementation. For a system owned set, the insertion class of the member records is always declared as automatic.

For every record obtained by applying mapping rules, there will be a corresponding record in the VAX-11 DBMS schema. The descriptions of the field items are derived from the same model. The area within which the record lies, will be the same as the data area of the entity in the E-R model.

For every set in the general CODASYL model, there will be a set defined in the VAX-11 DBMS. The owners and members of all the sets and the insertion/retention classes of the sets are obtainable from the general CODASYL model. To define the system-owned sets, it is necessary to study the labelling of the global functional entity model.

For every entity in the functional entity model, where the traversal frequency of the identifier edge is greater than the traversal frequency of any other property edge or relationship edge for that entity, a system-owned set is defined with the particular entity type as the member record of the set. The insertion class of the member record is defined as automatic. This is a requirement of a system-owned set. The retention class of the member record is defined as fixed. Once inserted, a record of this type will always be a member of the system-owned set, unless it is deleted from the data base.

The schema description also contains information about the ordering of a set. The ordering of a set depends on the usage pattern of each of the individual members. The insertion and retrieval conditions needs to be analysed. The functional analysis carried out for this project does not give enough information to draw any conclusion on the ordering of a set and thus the DBMS was allowed to assign a default value. This is an area which needs further study and the functional analysis phase needs to be refined and extended to capture enough information so that such decisions may be taken.

9.4.2 Implementation of the storage schema

The options for three possible storage structures, which were discussed in section 7.3, are available in the VAX-11 DBMS. Records which are associated with another record can either be clustered together or clustered around the record with which it is associated. In situations when neither of these options is applicable, a record may be inverted on a particular data item.

As stated above, our first preference is to place records around the associated record. Our second preference is to cluster records which share a common attribute or are related to another common record. The least preferred option is to invert a record on an attribute.

In this implementation, the clustering effect can be achieved by using the 'clustered via' option. This option can only be applied to records which are members of a specific set where the insertion class of the member record is automatic. The member records will be well-placed near the owner record if there is room in that storage page, otherwise they will be placed in a page as close as possible to that of the owner record. This cannot be achieved for records which are not members of sets for which the insertion class is not automatic.

The option of scattering records by hashing them on a data item, can be used to cluster records which share a common attribute. If this option is applied to data items with unique values, then the records will be scattered in the storage area and can be accessed directly if the value of the data item is provided. On the other hand, if the records are hashed on data items which are not unique, then all the records which have the same hash keys will be stored in the same page, thus achieving the effect of clustering.

Member records which do not have an automatic insertion class cannot be clustered near the owner record. But this does not produce any major problems. Entities which are accessed frequently from another entity, through a relationship, usually have obligatory membership in that relationship. If there are situations where entities which have no obligatory membership of a relationship are accessed through that particular relationship, then the only option available is to sort the member records by their keys and adopt an indexed set structure. This option is also appropriate to any set structure for which the other options are not applicable.

To be able to allocate the storage structure to each individual record and set, we need to examine the list of sets and records and the labelled functional entity model. For any record, if an edge labelled 'W' or 'C' is the identifier edge of the record, then the calc mode can be used. If the edge is a property edge, then the record can be scattered on that attribute or data item. If the edge is a relationship edge, then the records are clustered round the owner record via that set. For sets represented by an edge other than 'W' or 'C', an index mode is assigned.

The above discussion can be summarised by the following algorithm.

```
FOR every record represented by a node,

    IF the edge labelled 'W' or 'C'
         is the identifier edge
        THEN assign the calc mode;

    ELSE IF it is a property edge
        THEN scatter on that attribute

    ELSE it is a relationship edge
        cluster the records via that set.
```

```
FOR every set represented by an edge

    IF the edge is labelled 'W' or 'C'
        THEN the members will be clustered
            via that set

    ELSE the edge will be labelled 'I'
        assign index mode of storage.
```

The DBMS assigns a default storage structure to all the records and sets. The DBMS assigns the default values depending on the schema definition. The storage schema obtained by applying the above algorithm is based on the usage pattern of the data. It provides the data base designer with some guidance on how to tune the storage schema, so as to give a better performance.

9.5 Conclusion

In comparison with the relational implementation, the CODASYL implementation is far more complicated. It is difficult to automate much of the process of CODASYL implementation. To be able to define the CODASYL schema and the storage schema automatically, additional information has to be incorporated in our data and functional analysis phase. Extensive investigation is needed before we can produce a complete CODASYL schema and storage schema. The work presented in this research could produce a first-cut definition of these, which would then need to be modified when a better understanding of the usage pattern is gained.

The implementation of the CODASYL mappings was not part of the remit of this research. However, most activities are orientated toward relational DBMS and these are likely to be most widely used in the future as they are flexible and provide a simple interface to users. However, the discussion in this chapter illustrates that a CODASYL implementation can be automated in the same way as the relational mappings.

CHAPTER 10

CONCLUSION

10.1 Introduction

The main objective of the research was to develop software tools for automatically converting the analyst's data model into a information system on a particular DBMS. The previous chapters describe techniques, which were used to fulfil the stated objective. In this chapter the the reasons for selecting the specific programming language for software development, the contributions of the research work and areas for further development are reviewed.

10.2 Selection of the programming language

All the programs are written in standard PASCAL. PASCAL was chosen, particularly, because of its rich set of data types and its portability. The decision was influenced by the fact that:-

- it allows systematic program development,
- it allows flexible data structures to be implemented efficiently,

- it allows extensive error checking facilities because it is a strongly-typed language, that is, every defined data object belongs to unique type,
- it is becoming popular as a programming language for microcomputers.

The programs were first developed on a PRIME machine. It was possible to transfer the programs on to a VAX machine and run them successfully without many modifications, though the two machines run different versions of PASCAL.

10.3 Contribution

The contribution of this research can be categorised into three main areas:

i. The development of software tools for logical mapping. This automatically converts the result of the data analysis, i.e., the conceptual model, to the logical model of relational and CODASYL types of DBMS;

ii. The development of software tools for physical design. This analyses the result of functional analysis and suggests ways of structuring the data depending on the usage pattern; and

iii. The development of software tools which will automatically (partly) implement the chosen data model on to a selection of DBMS, on the basis of the results of an analysis of the physical and logical characteristics of the DBMS.

Each of these contributions will be discussed in turn.

10.3.1 Logical mapping

The software written for this area examines the semantic content of an E-R data model and transforms it into a relational data model. The relational model is such that there are no redundant relations and all the relations are in fourth normal form.

This software can be used as a tool by a data base designer to derive the relational or CODASYL logical model from the conceptual model. It saves the designer from some of the tedious manual processes of going through every entity and relationship in the model and mapping it to the relations for the relational model and, similarly, the set and record structures for the CODASYL model. This is not dependent on any particular implementation, so its relevance is not restricted to the DBMS used by the author.

The mapping of the E-R model on to the relational model is straightforward. The resultant relational model contains normalised relations which are free of any update anomalies. However, a relational model has no features which could describe the obligatory/non-obligatory membership of entities in an E-R model. Therefore, update operations which arise as side effects of another update operation will not be performed automatically. Once this logical model is implemented on any target system, special software has to be developed which will deal with these update operations.

The mapping of the E-R model on to the CODASYL model is not as straightforward as the relational model. Problems arise because the logical model definition of a CODASYL DBMS contains features which describe how data should be physically organised. For the logical mapping, only those features of the CODASYL recommendations which drive the logical mapping were chosen. The CODASYL model supports all the features of the data portrayed by the E-R model, including the membership class of entities. Though the insertion/retention class of a member of a set describes the membership class of an entity, it is not possible to assign the insertion/retention class accurately from the E-R model. A thorough functional analysis which analyses the entity life history (that is the various states through which an entity progresses from the time it is created until the time it is deleted) is necessary to be able to define these accurately.

10.3.2 Physical design

The software designed for this phase examines the known functions for the area under investigation and analyses the usage pattern of the data. It provides the data base designer with a detailed knowledge of how an entity is being accessed and how frequently it is being accessed. The use of this software is not restricted to any particular implementation. It frees the data base designer from the tedious task of deciding how each attribute of every entity and each relationship is being used by the different functions. The software also suggests the access mechanism to be assigned to each of the access paths. The information can be used to select the actual implementation structure supported by specific DBMS.

The results obtained from the physical design give a clear understanding of the storage structure necessary. This information can be used in deciding the kind of support we like to have when choosing our target DBMS. If the target system is already known, then this would enable the designer to decide how the physical characteristics of the target system can be exploited to give the kind of support he would like to assign to each access path.

When implementing the model on INGRES, it is found that INGRES supports clustering and indexing, but it is not possible to place tuples from different relations near each other. In MIMER, only secondary indexing can be added to enhance performance. As data is stored in a B*-tree structure, it cannot support 'clustered' or a 'well-placed' structure. In VAX-11 DBMS there are features which can be exploited, so that all three structures 'indexed', 'clustered' and 'well-placed', can be supported.

10.3.3 Implementation

The use of the software developed for implementation is restricted to the DBMS selected by the author. The physical characteristics supported by different DBMS vary. The interfaces for different DBMS are different. Therefore, it is not feasible to develop general software that would enable a data base designer to implement a data base on to any target DBMS. The programs written for this phase are driven by the physical characteristics and the interface supported by the target DBMS. However, the software illustrates that the implementation of a data base on to target DBMS chosen is a logical process which can be partly automated if the logical and physical characteristics of the data to be implemented are known.

10.4 Future work

The major goal of this research was to develop software tools to data base design methodology. Tools and techniques were developed which make a positive contribution to the achievement of this goal. In order to fully automate the process further work needs to be done in the following areas.

One possibility is to develop similar tool for micro-computers. A portable software package, named Database Implementation Made Easy (DIME), based on a micro-computer, is currently being developed, see Chaudhuri and Esendal (1986).

DIME is a software tool which supports data base design and implementation. It is developed on IBM PCs. It documents the results of data analysis and functional analysis and produces logical model definitions for relational and CODASYL DBMS independent of any particular implementation. Furthermore, it examines the results of functional analysis and advises designers on ways of organising the data to enhance the performance of the prospective data base. The package can be used as an aid to data base design, independently of any particular DBMS implementations. However, the authors have developed software to interface DIME with INGRES running on a VAX-750 under a UNIX operating system.

Various microcomputer-based data base design tools are commercially available. Generally, they offer a good graphic interface for the designer to communicate with the system and most of them claim to produce first cut data base design models. However, most of them are simply tools for documenting and checking the consistency of the results of the analysis. Some of them produce a relational or CODASYL structure but they are only a first cut model. DIME does not provide any graphic interface, but as a design tool it goes a lot further than these tools. Not only does it document and check the consistency of the results of data analysis, it produces optimised relational and CODASYL structures and analyses the usage pattern of the data and advises the designers on how to physically organise the data.

It is not possible to prove that the algorithm used to assign support to the access paths does produce an optimal solution. However, it can be stated that it is a good solution if reducing access time is of prime importance.

In the realm of functional analysis, several problems and deficiencies still exist. The details gathered from this phase were insufficient to take certain decisions in a later phase, namely, accurately assigning the insertion/retention class of the CODASYL sets. Entity life history analysis may have potential in giving a better logical and physical design. A full study is needed to explore this hypothesis.

The conceptual model used in this project is derived from an analysis of the user requirements by a data analyst. Advances in artificial intelligence and expert systems research and development make it feasible to consider sophisticated tools to help in this analysis, and may eventually be able to produce the conceptual model without a data analyst.

APPENDIX A

Data Models for the Hospital

Pathology Laboratory.

The Pathology Laboratory is concerned with biochemistry, haemotology and microbiology activities. The information flow within this area is concerned with the investigation and reporting of the causes and manifestation of diseases by using chemical, microscopic, biological and bacteriological methods.

The Pathology Laboratory can be further partitioned into a few smaller departments according to the type of job they do.

(1) Serology department: This department gets a medical request for blood for a certain patient. The request is filed. It then registers the patient's data (blood group, etc.). A record is kept of all patients with an antibody problem. The blood report, and any important information which should be stored, is also stored in the patient's record. This is held in the record office.

(2) Protein department: This department keeps a day book, that is, details about all patients who have been tested on that day. The department keeps a record of any interesting or unusual cases. It also keeps a reference of all the patients sent to another specialised hospital.

(3) Anti coagulant clinic: This department is responsible for patients attending anti coagulant clinics. They keep a record of the result and the dose of the drug used for the six most recent visits. The patients are sent to this department by their own doctor. Patient details are formed from it. The entity model for this department is shown in Figure A.1.

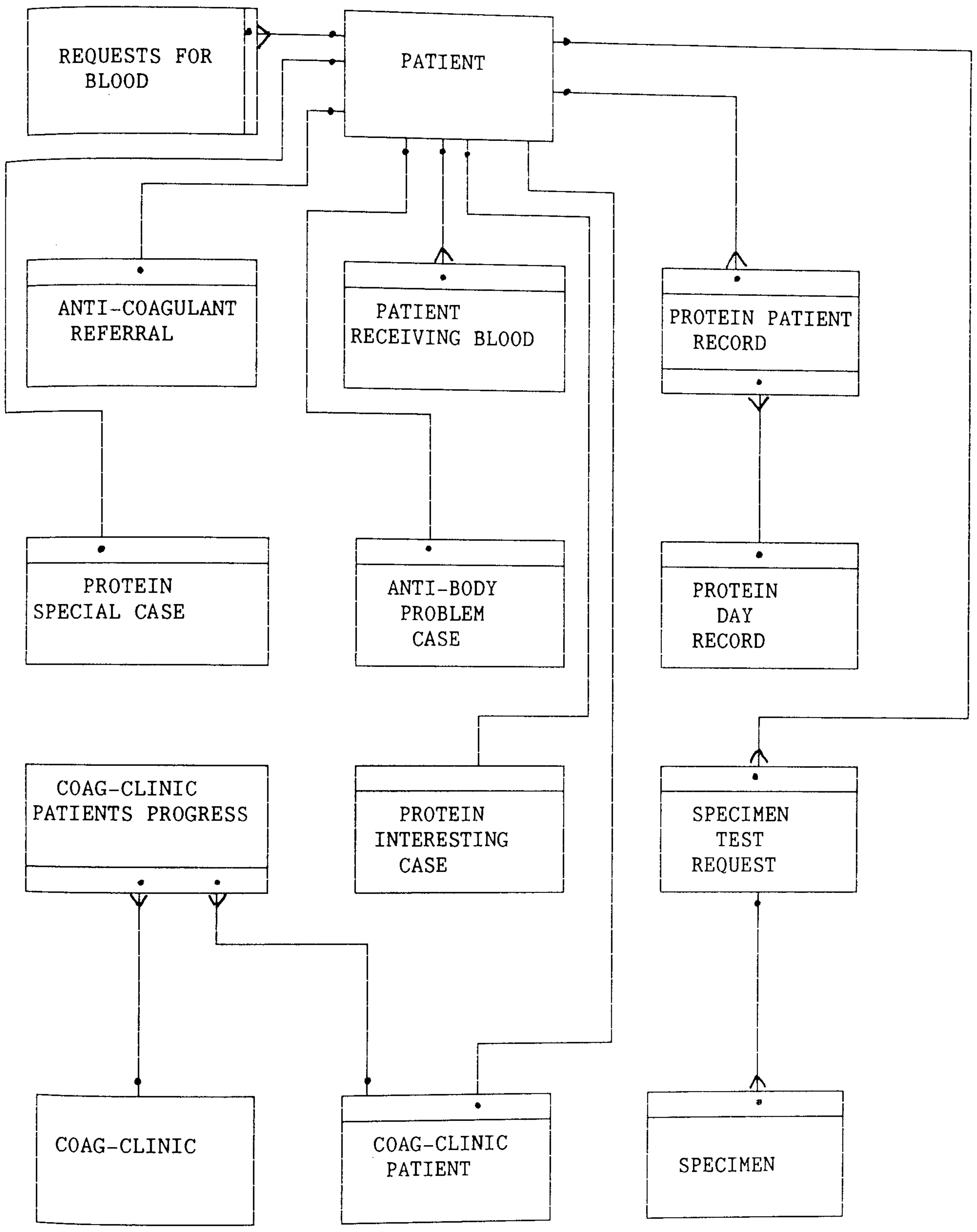


Figure A.1 Entity Model for Pathology Laboratory

Pharmacy Department

The Pharmacy Department is concerned with the activities of purchasing, testing, storing and dispensing drugs. This department is responsible for processing the prescription, drug ordering, drug records and stock control, drug administration and statistics relating to hospital prescribing.

The entity model for this department is as Figure A.2.

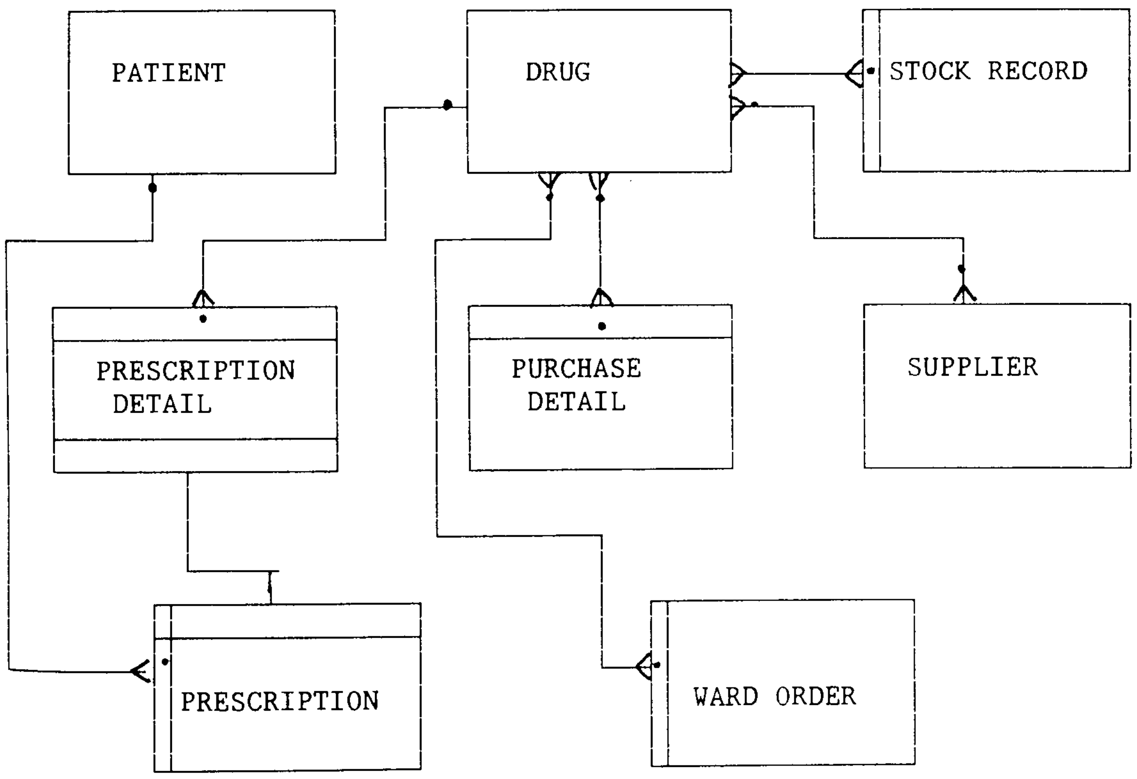
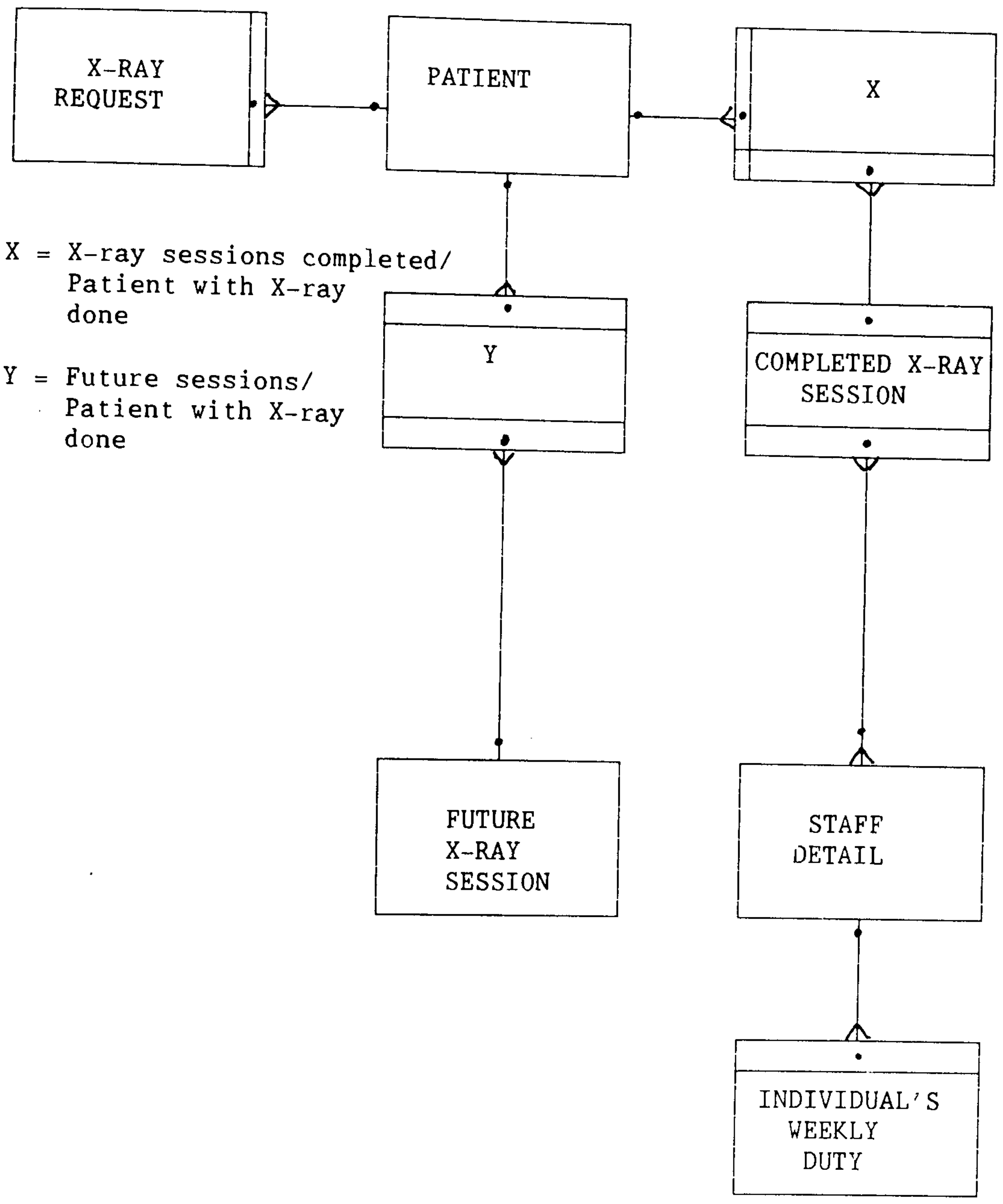


Figure A.2 Entity Model for Pharmacy Department

X-ray Department.

The X-ray department's main functions are the production and interpretation of and reporting on, X-ray films of patients taken at the request of a doctor in the hospital or outside. The information processes that are involved are the work schedule in the department and the request to the radiology sector. The entity model formed is as Figure A.3.



X = X-ray sessions completed/ Patient with X-ray done

Y = Future sessions/ Patient with X-ray done

Figure A.3 Entity Model for X-ray Department.

The Wards

The wards are responsible for caring for the patients who come in for a long or a short period. They are responsible for providing bed-state information and keeping track of all the medications, treatment and progress of each patient during his stay in the hospital. The entity model for this data area is as shown in Figure A.4.

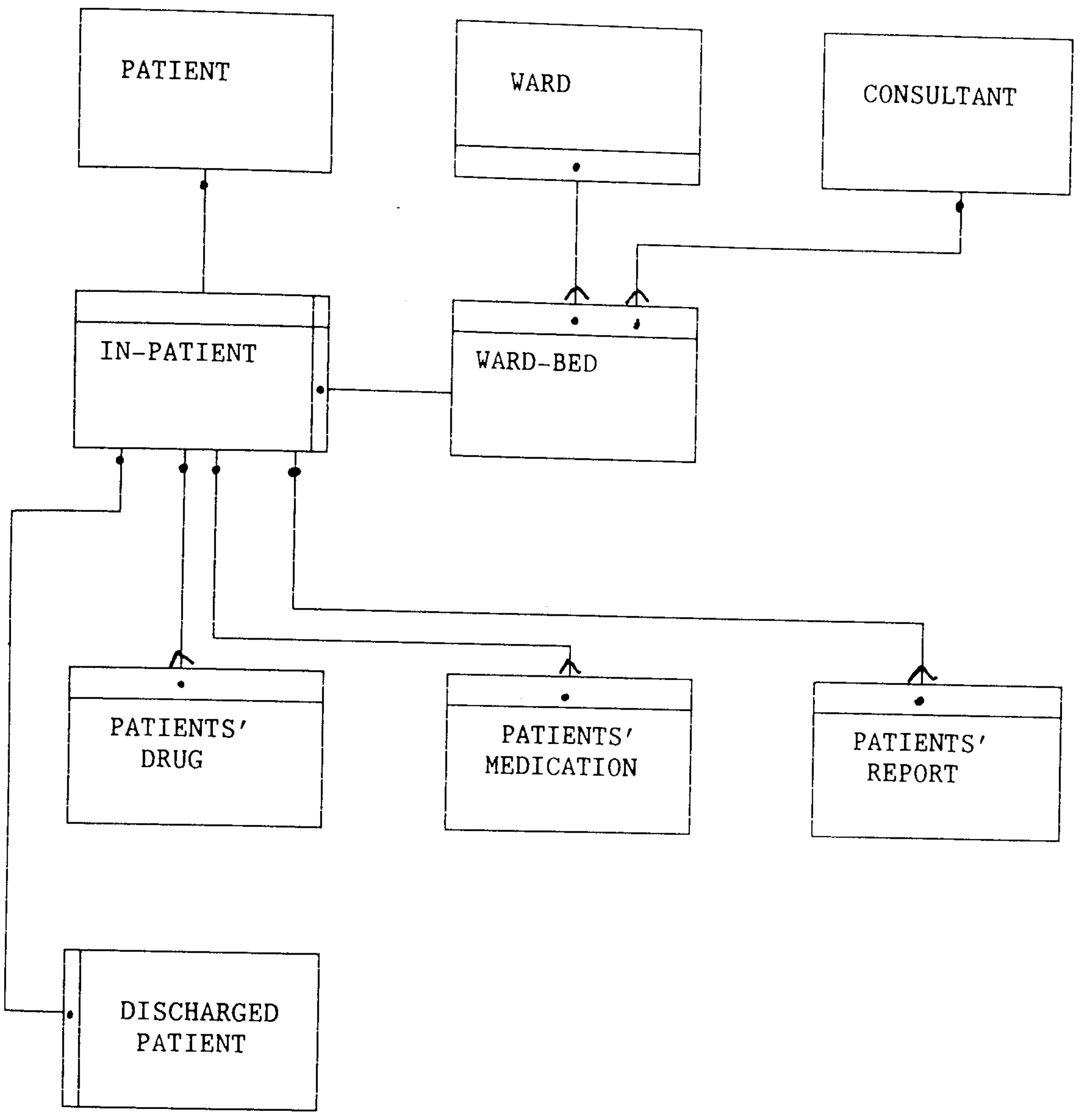


Figure A.4 Entity Model for the Wards.

Nursing Department.

The other area analysed was that of the nursing staff. The primary objective of this department is to manage nursing resources effectively. This department is responsible for keeping records of all current nursing staff and also those of nurses who have worked at the hospital. This department is also responsible for monitoring the nurses' sickness record, their holidays and duty rota. The entity model is as Figure A.5.

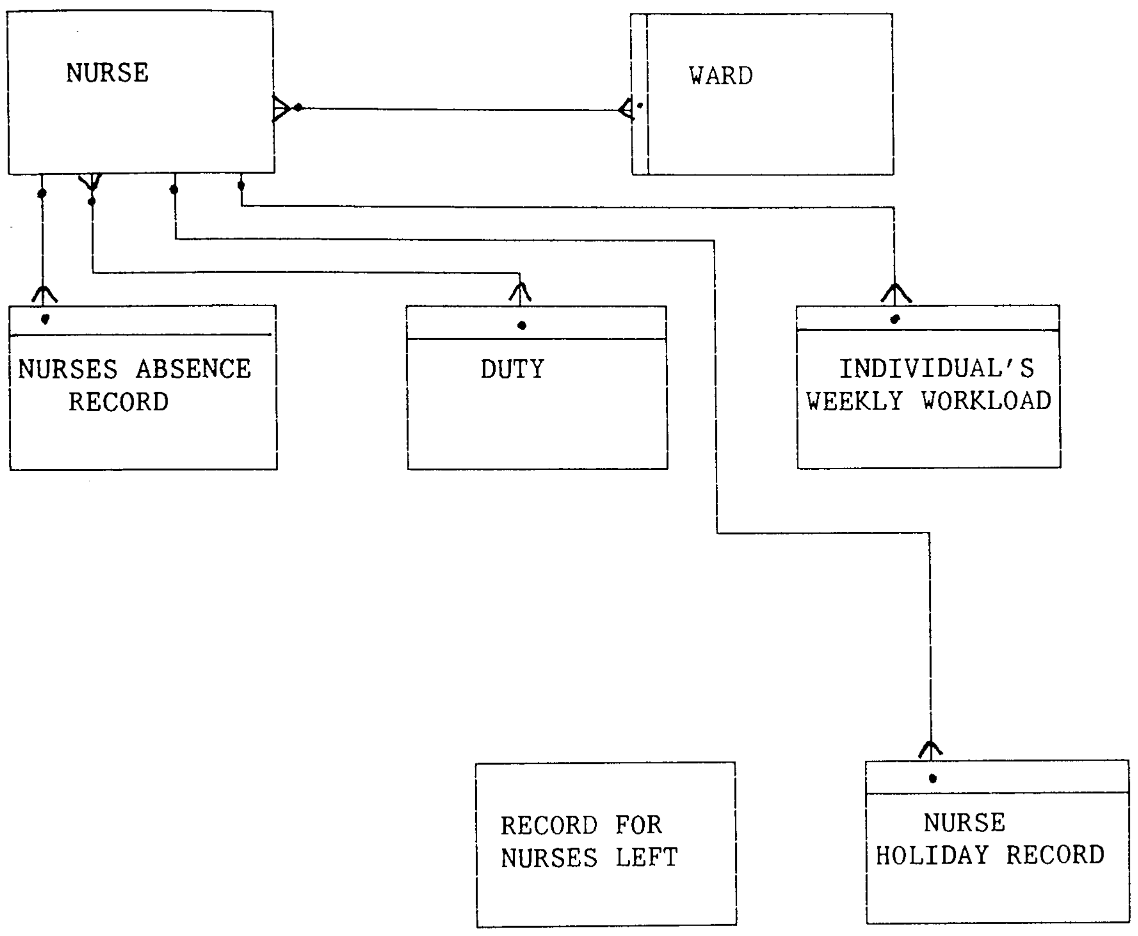
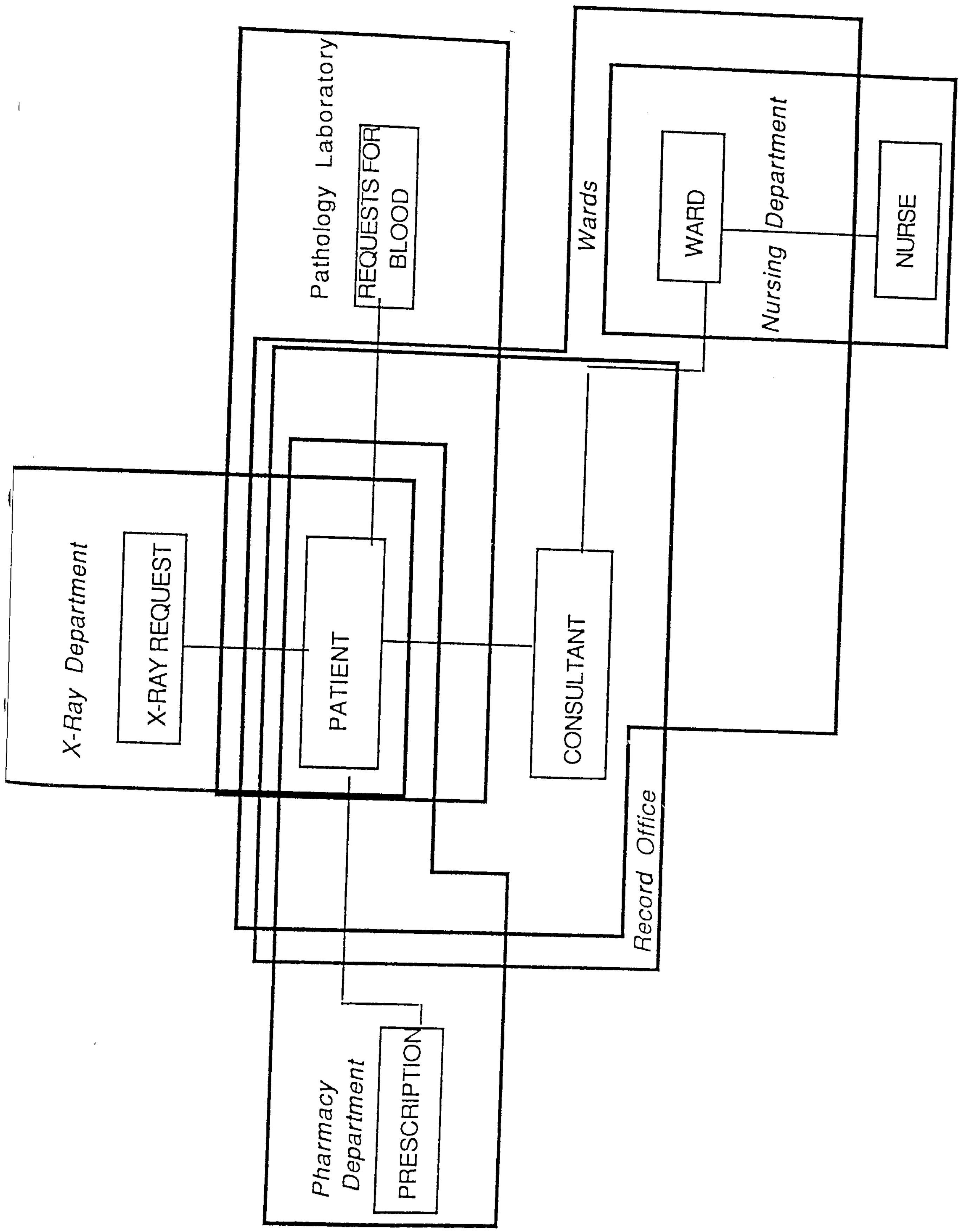


Figure A.5 Entity Model for the Nursing Department.

APPENDIX B

Figure B.1 Merged Data Model for the Hospital



REFERENCES

ANSI/X3/SPARC DBMS Framework (1977), **Interim report: Study group on data base management systems.**
AFIPS Press, 1977.

ANSI/X3/SPARC/STUDY GROUP DATABASE MANAGEMENT SYSTEMS, (1978), **"The ANSI/X3/SPARC DBMS Framework".**
Information Systems, 3, 3, 1978, pp 173-191.

ANSI X3H2 (1981), **"Proposed American National Standard for a definition Language for Network Structured Databases".**
American Structured Standards Institute, 1981.

Atre, S., (1980), **"Data Base, Structured Techniques for Design, Performance and Management".**
Wiley and Sons, New York: 1980.

Avison, D.E., (1985), **"Information Systems Development: A Data Base Approach".**
Blackwell Scientific Publications, U.K., 1985.

Bachman, C.W., (1969), **"Data Structure Diagrams".**
Database, 1 2, 1969, pp 4-10

Bachman, C.W., Daya, M.,(1977), **"The Role Concept in Data Models".**
Proceedings of the 3rd Conference on Very Large Data Bases, ACM, 1977.

Bader, J., Hannaford, D., Cochran, A., Edwards,J., (1987), **"Intellipse: A Knowledge Based Tool for an Integrated Project Support Environment".**
Proceedings of the Conference on Automating Systems Development, Leicester, April, 1987.

Benci, E., Bodart, F., Bogeart, H., Cabanes, A., (1976) **"Concepts for the Design of a Conceptual Schema".**
Proceedings of the IFIP-TC-2 Working Conference on Modelling in Data Base Management Systems, North-Holland, Freudenstadt, January 1976.

Boehm, B.W., (1976), **"Software Engineering".**
TRW Tech. Report TRW-SS-76-98, October, 1976.

Bracchi, G., Paolini, P., Pelagatti, G., (1976) **"Binary Logical Associations in Data Modelling".**
Proceedings of the IFIP-TC-2 Working Conference on Modelling in Data Base Management Systems, North-Holland, Freudenstadt, January 1976.

Braegger, R.P., Dudler, A.M., Rebsamen, J., Zehnder, C.A., (1985), **"Gambit: An Interactive Database Design Tool for Data Structures, Integrity Constraints and Transactions".**
IEEE Transaction Software Engineering (USA) **11**, 7, July 1985, pp 574-583.

Brown, P.J., (1982), **"Tools for Amateurs".**
Tools and Notions for Program Construction, Cambridge University Press, 1982.

Bubenko, J.A. (Jr), (1977) **"The Temporal Dimension in Information Modelling".**
Proceedings of the Conference on Architecture and Models in Data Base Management Systems, North-Holland, Amsterdam, 1977.

CACI (1980) **"Analysis of the Data Resource".**
Internal CACI Training Course.

Chaudhuri, J., Esendal, H.T., (1986) **"A Preprocessor Package to Facilitate Data Base Design".**
Fourth IASTED International Symposium on Applied Informatics, Innsbruck, 1986.

Chen, P.P.S., (1976), **"The Entity Relationship Model- Towards a Unified View of Data".**
ACM Transactions on Database Systems, 1 ,1, March 1976, pp 9-36.

Chen, P.P.S., (1977), **"The Entity-Relationship Approach to Logical Data Base Design".**
Q.E.D. Monograph Series, Wellesley, MA, 1977.

Chen, P.P.S., Yao, S.B., (1977), **"Design and Performance Tools for Database Systems".**
Proceedings of the Third International Conference on Very Large Data Bases, ACM, 1977.

Chen, P.P.S., (1980), **"E-R Approach to Systems Analysis and Design".** Proceedings of the International Conference on the Entity-Relationship Approach to Systems Analysis and Design, North Holland, Amsterdam, 1980.

Clarke, J.D., Hoffer, J.A., (1979), **"Physical Database Record Design".**
Q.E.D. Information Systems, Wellesley, MA, 1979.

CODASYL Development Committee (1962), **"An Information Algebra".**
CACM, April 1962, pp 190-204.

CODASYL (1971), **"Database Task Group of CODASYL Programming Language Committee Report."**
ACM, New York, USA, April 1971.

CODASYL (1978), **"CODASYL Data Description Language Journal of Development".**
Material Data Management Branch, Dep. of Supply and Services, Ottawa, 1978.

Codd, E.F., (1970) **"A Relational Model of Data for Large Shared Data Banks".**
CACM, **13**, 6, June 1970, pp 377-387.

Comer, D., (1978), **"The Ubiquitous B-Tree."**
ACM Computing Survey, **11**, 2, June, 1979, pp 121-137.

Crocker, P.S., (1984), **"Systems Analysts-Training and Experience".**
National Computing Centre, Manchester, 1984.

Date, C.J., (1986), **"An Introduction to Database Systems"** Vol 1.
Fourth Edition. Addison Wesley, London, 1986.

Davenport, R.A., (1978) **"Data Analysis for Database Design".**
Australian Computing Journal, 10, 4, November, 1978, pp 122-137.

Davis, G.B., Olson, M.H. (1985), **"Management Information Systems: Conceptual Foundations, Structure, and Development."**
Second Edition, McGraw-Hill Book Company, New York, 1985.

Davis, K.H., Arora, A.K., (1985), "**August: An Experimental System to Translate a Conventional File System into a Commercial Database System**".
Proceedings of COMPSAC 85, IEEE Computer Society Press, 1985.

DeMarco, T., (1978), "**Structured Analysis and System Specification**".
Yourdon, Inc., New York, 1978.

Digital Equipment Corporation (1981a), "**VAX-11 DBMS V1 DDL Reference Manual** ".
Digital Equipment Corporation, Maynard, Massachusetts, 1981.

Digital Equipment Corporation (1981b), "**VAX-11 DBMS V1 Summary Description**".
Digital Equipment Corporation, Maynard, Massachusetts, 1981.

Falkenberg, E.,(1976), "**Concepts for Modelling Information**".
Proceedings of the IFIP-TC-2 Working Conference on Modelling in Data Base Management Systems, G.M. Nijssen (ed), North-Holland, Freudenstadt, January 1976.

Ferrara, F.M., Batini, C., "**Practical Application of Idef1 as a Database Development Tool**".
Database (USA), **15**, 4, 1984, pp 15-20.

Finkelstein, C., (1980), "**Data Analysis and Design of Information Systems**".
Infocom, Australia, 1980.

Fry, J.P., Deppe, M.E., (1976), "**Distributed Data Bases: A Summary of Research**".
Computer Networks, **1**, 2, 1976, pp 1-13.

Fry, J.P., Sibley, E.A., (1976), "**Evolution of Database Management Systems**".
Computing Surveys, **8**, 1, 1976, pp 7-42.

Gane, C., Sarson, T., (1979), "**Structured System Analysis**".
Prentice-Hall, Inc., Englewood Cliffs, NJ, 1979.

Gerritsen, R., (1975), "**A Preliminary System for the Design of DBTG Data Structures**".
Communication of ACM, **18**, 10, 1975, pp 557-567.

Gerritsen, R., (1978), "**Steps Towards the Automation of Database Design**".
NYU Symposium on Database Design, May 1978, pp 91-99.

Gray, P., (1984), "**Logic, Algebra and Databases**".
Ellis Horwood Ltd., Chichester, 1984.

Grotenhuis, F., (1976), "**A Conceptual Model for Information Processing**".
Proceedings of the IFIP-TC-2 Working Conference on Modelling in Data Base Management Systems, G.M. Nijssen (ed), North-Holland, Freudenstadt, January 1976.

Hall,P., Owlett,J., and Todd,S. (1976), "**Relations and Entities**".
Proceedings Ofthe IFIP-TC-2 Working Conference on Modelling in Data Base Management Systems, G.M. Nijssen (ed), North-Holland, Freudenstadt, January 1976.

Hein, K.P., (1985), "**Information System Model and Architecture Generator**".
IBM System Journal, **24**, 3, 1985, pp 213-235.

Hoffer, J.A., (1975), "**A Clustering Approach to the Generation of Subfiles for the Design of a Computer Database**".
Ph.D. Thesis, Department of O.R, Cornell University, 1975.

Hoffer, J.A., Severance, D.G., (1975), "**The Use of Cluster Analysis in Physical Data Base Design**".
Proceedings of the First International Conference on Very Large Databases, ACM, 1975.

Howden, W.E., (1982), "**Contemporary Software Development Environments**".
Communications of the ACM, **25**, 5, May 1982, pp 318-329.

Howe, D.R., (1983), "**Data Analysis for Data Base Design**".
Edward Arnold Ltd., London, 1983.

Inmon, W.H., (1981), "**Effective Data Base Design**".
Prentice-Hall, Inc., Englewood Cliffs, N.J., 1981.

Kahn, B.K., (1976), "**A Method for Describing the Information Required by the Data Base Design Process**".
Proceedings of the International ACM/SIGMOD conference on Management of Data, 1976.

Kahn, B.K., (1978), "**A Structured Logical Data-Base Design Methodology**".
NYU Symposium on Database Design, May 1978, pp 15-24.

Kent, W., (1978), "**Data and Reality: Basic Assumptions in Data Processing Reconsidered**".
North-Holland, Amsterdam, 1978.

Kerschberg, L., et al., (1976), "**A Taxonomy of Data Models**".
Tr. Csrb-70, Computer Systems Research Group, University of Toronto, 1976.

Kim, W., (1979), "**Relational Database Systems.**"
ACM Computing Surveys, **11**, 3, September 1979, pp 185-211.

King, P.J.H., (1977), "**Information Analysis for Data Base Design**".
On-line 1977, Data Base - London, On-line Conferences Ltd, 1977.

King, R., McLeod, D.,(1985), "**A Database Design Methodology and Tool for Information Systems**".
ACM Transactions on Information Systems, 3, 1, 1985.

Kroenke, D., (1983), "**Database Processing**".
Second Edition, Palo Alto, CA: Science Research Associates, Inc., 1983.

Langefors, B., (1974), "**Information Systems**".
Proceedings of International IFIP Congress, Amsterdam, 1974.

Lum, V.Y., (1979), "**1978 New Orleans Data Base Design Workshop Report**".
Procedings of the Fifth International Conference on Very Large Data Bases, ACM, October, 1979.

Lusk, E.L., Overbeek, R. A., (1981), "**A Practical Design Methodology for the Implementation of IMS Database using Entity-Relationship Model**".
ACM-Sigmod International Conference on Management of Data, Santa Monica, 1981.

Martin, G., (1985), **"System Design from Provably Correct Constructs".**
Prentice-Hall, Inc., Englewood Cliffs, N.J., 1985.

Martin, J., McClure, C., (1985), **"Diagramming Techniques for Analysts and Programmers".**
Prentice-Hall, Inc., Englewood Cliffs, N.J., 1985.

McGee, W.G., (1976), **"On User Criteria for Data Model Evaluation".**
ACM Transaction on Data Base System, 1, 4, December 1976, pp 370-387.

Mealy, G.H., (1976), **"Another Look at Data".**
Proceedings of the AFIPS Fall Joint Computer Conference, 1976, 31.

Moulin, P., Randon, J., Teboul,M., Savoysky, S., Spaccapietra, S., Tardieu, H., (1977), **"Conceptual Model as a Data Base Design Tool".**
Proceedings of the Conference on Architecture and Models in Data Base Management Systems, North-Holland, Amsterdam, 1977.

Navathe, S.S., Gadgil, S.G., (1980), **"A Methodology for View Integration in Logical Database Design".**
Database Systems Research and Development Center Tech. Report, University of Florida, 1980.

Navathe, S.B., Schkolnick, M., (1978), **"View Representation in Logical Database Design".**
Proceedings of ACM SIGMOD International Conference on Management of Data, 1978.

NCC, (1980), **"Evaluation and Implementation of Database Systems".**
Database Design Tools, NCC, Manchester, 1980.

Novak, D., Fry, J., (1976), **"The State of the Art of Logical Database Design".**
Proceedings of 5th Texas Conference on Computer Systems(IEEE), 1976.

Olle, T.W., (1978), **"The Codasyl Approach to Data Base Management".**
Wiley Interscience Publication, New York, 1978.

Orman, L., (1984), **"Familial Model of Data".**
International Journal of Computing and Information Science. 13, 3, June 1984, pp 149-175.

Orr, K.T., (1976), "**Structured Systems Design**".
Langston and Associates, Topeka, Kansas, 1976.

Palmer, I., (1978), "**Practicalities in Applying a Formal Methodology to Data Analysis**".
Proceedings of NYU Symposium on Database Design, 1978.

Parkin, A., Thornton, S.R., Holley, P.J., (1987), **"Can-fact-finding-be-automated?"**
Conference On Automating Systems Development, Leicester Polytechnic, April, 1987.

PCTE, (1986), "**PCTE-A Basis for a Portable common Tool Environment; Functional Specification**".
Bull, GEC, ICL, Nixdorf, Olivetti, Siemenns, 1986.

Relational Technology Inc.(1983), "**INGRES Reference Manual, Version 1.4.**".
Berkeley, CA, 1983.

Robinson, H., (1985), "**Database Analysis and Design**".
Chartwell Bratt, Bromley, 1985.

Rock-Evans, R., (1981), "**Data Analysis**".
A Computer Weekly Publication, IPC Business Press, 1981.

Sakai, H., (1980), "**Entity Relationship Approach to the Conceptual Schema Design**".
ACM Sigmod, International Conference on Management of Data, May 1980.

Savant Enterprise (1985) "**MIMER , Concpts and Facilities** ".
Savant Enterprises, Carnforth, 1985.

Schmid, H.A., (1977), "**An Analysis of Some Contents for Conceptual Models**".
Proceedings of the Conference on Architecture and Models in Data Base Management Systems, North-Holland, 1977.

Senko, M. E., Altman, E.B. Astrahan, M.M., Fehder, P.L., (1973), "**Data Structures and Accessing in Data-Base Systems**".
IBM Systems Journal, **12**, 1, 1973, pp 30-93.

Severance, D.G., and Carlis, J.V., (1977) **"A Practical Approach to Selecting Record Access Paths".**
ACM Computing Surveys, **9**, 4, 1977, pp 259-272.

Stucki, L.G., Walker, H.D., (1981), **"Concepts and Prototypes of ARGUS".**
Software Engineering Environments, North Holland, 1981, pp 61-79.

Sundgren, B., (1978), **"Data Base Design in Theory and Practice - Towards an Integrated Methodology".**
Proceedings of the 4th Conference on Very Large Data Bases, ACM, 1978.

Sutcliffe, A.G., Layzell, P.G., Loucopoulus, P., Davis, C. G., (1987), **"Majic - Automating JSP Program Design and Dvelopment".**
Procedings of the Conference on Automating Systems Development, Leicester Polytechnic, April, 1987.

Teorey, T.J., Fry, J.P., (1982), **"Design of Database Structures".**
Prentice-Hall, Inc., Englewood Cliffs, N.J., 1982.

Tozer, E.F., (1976), **"Data Systems Analysis and Design".**
Proceedings of the Conference European Coop. Informatics, 1976.

Tsao, J.H., (1980), **"Enterprise Schema: An Approach to IMS Logical Data Base Design".**
ACM-Sigmod International Conference on Management of Data, May 1980.

Tsichritzis, D.C., Lochovsky, F.H., (1982), **"Data Models".**
Prentice-Hall, Inc., Englewood Cliffs, N.J., 1982.

Ullman, J.D., (1982), **"Principles of Database Systems".**
Computer Software Engineering Series, Computer Science Press, Inc, Maryland, 1982.

Wasserman, A.I., (1980), **"Information System Design Methodology".**
Information Science Journal, **31**, 1, 1980.

Williams, M.H., Pattison, I.M., Nerves, J.C., (1986), **"Reorganisation in a Simple Database System".**
Software Practice and Experience, **16**, 8, 1986.

Yao, S.B., (1977), "**An Attribute Based Model for Database Access Cost Analysis**".
ACM Transaction on Database Systems, 2, 1, 1977, pp 45-67.

Yao, S.B., De Jong, D., (1978), "**Evaluation of Database Access Paths**".
Proceedings of the ACM/SIGMOD International Conference on Management of Data, 1978.

Yourdon, E., Constantine, L.L., (1979), "**Structured Design**".
Prentice Hall, Inc., Englewood Cliffs, NJ, 1979.