Facilitating Evolution in Relational Database Design:

A procedure to evaluate and refine novice database designers' schemata

A thesis presented in partial fulfilment of the requirements for the degree of

Master of Business Studies in Information Systems

at Massey University

Michael Robert Ryder

1996

Acknowledgments

I feel the desire to thank several people for the support they have shown me throughout the duration of this research.

Many thanks go to Clare Atkins for continually sparking my enthusiasm for issues surrounding database and data modelling, for acting as a second researcher and for her unyielding support and proofreading of this thesis. Also to Wen van Kersbergen from the Amsterdam School of Business for his expert knowledge and assistance.

Thanks also go to Jon Patrick for his advice, guidance and support throughout the process of conducting this research. To the survey respondents and Angus, I also extend my gratitude.

To the staff of the Information Systems Department, thank you for your interest in the thesis's progress and your patience whilst sitting through the seminars it has given rise to.

Lastly, to my friends and family, your perseverance and patience and encouragement is greatly appreciated.

"A little learning is a dang'rous thing;
Drink deep, or taste not the Pierian spring:
There shallow draughts intoxicate the brain,
And drinking largely sobers us again."

(Alexander Pope)

Ars longa, vita brevis

Abstract

Relational database management systems (RDBMS) have become widely used by many industries in recent years. Latterly these systems have begun to expand their market by becoming readily available at minimal cost to most users of modern computing technology. The quality of applications developed from RDBMSs however is largely dependent upon the quality of the underlying schema.

This research looks at the area of schema design and in particular schemata designed by people who have a minimal understanding of relational concepts. It uses a survey and case studies to help define some of the issues involved in the area. A procedure to modify existing schemata is described, and the schema from one of the case studies used to apply the schema re-design procedure to a real database design. The results are compared to the original schema as well as a schema designed using a conventional application of the NIAM analysis and design methodology.

The research supports the hypothesis that database applications based on schemata designed by lay-persons are currently being used to support business data management requirements. The utility, reliability and longevity of these applications depend to some extent on the quality of the underlying schema and its ability to store the required data and maintain that data's integrity. The application of the schema re-design procedure presented in this thesis reveals refinements on the original schema and provides a method for lay-persons to evaluate and improve existing database designs.

A number of issues and questions related to the focus of this research are raised and, although outside the scope of the research, are noted as suggestions for further work.

Table of Contents

SECTION ONE

OVERVIEW

1 Introduction

People and organisations have had, and probably always will have, interests that require them to store, update, manipulate, query, summarise and retrieve data. In the past, resources dedicated to these processes have been proportionate to each person's means and requirements. People in large organisations tend to have relatively large data processing requirements and relatively large budgets so are likely to commit relatively large resources to the management of their data. Smaller organisations and individuals (known as the small office, home office environment (SOHO)) tend to have smaller budgets and less data to manage so consequently manage their data in ways more appropriate to the scope of their requirements. Where larger organisations have employed the skills of Information Systems professionals and database management systems, individuals and smaller organisations have tended to retain manual procedures and manual filing systems to manage their data (Batra et al, 1990).

Much of the literature concerning the use of database technology outside the auspices of Information Systems professionals concentrates on how to teach end users suitable techniques for data modelling ((Batra et al, 1990), (Davis et al, 1988), (Srinivasan, 1992)). The emphasis of these researchers' papers concerns the identification of strategies for enabling end users to get their database designs correct before they implement and use them. The present research takes the view that some people will ignore or be unaware of database design issues and therefore will design a poor quality database whose design will at some stage need to revised for it to fulfil a useful function.

For the purposes of this report it is desirable to closely define the characteristics of the type of person on whom the report focuses, the environment in which they are working and the tasks they may typically perform. This definition is included as a convenience for the purposes of this report rather than as a comprehensive and rigorous classification scheme. Figure 1 presents a breakdown of these characteristics showing

a three-dimensional box divided into six sections onto which three overlapping working environments have been placed creating five labelled working environments that permeate through the six sections.

The depth perspective of the block highlights that, at least for the purposes of this report, there is an important distinction between database designers and database users. The database user is defined here as the person who interacts with the data in the database, using the system's facilities for data entry, querying and reporting purposes. A database designer, on the other hand, is someone who interacts with the RDBMS to build and modify the database structure. It is possible in some circumstances that the database user may also be the database designer. In this case the distinction between the two is unimportant for this report, and the term designer will still be used to denote this type of person.

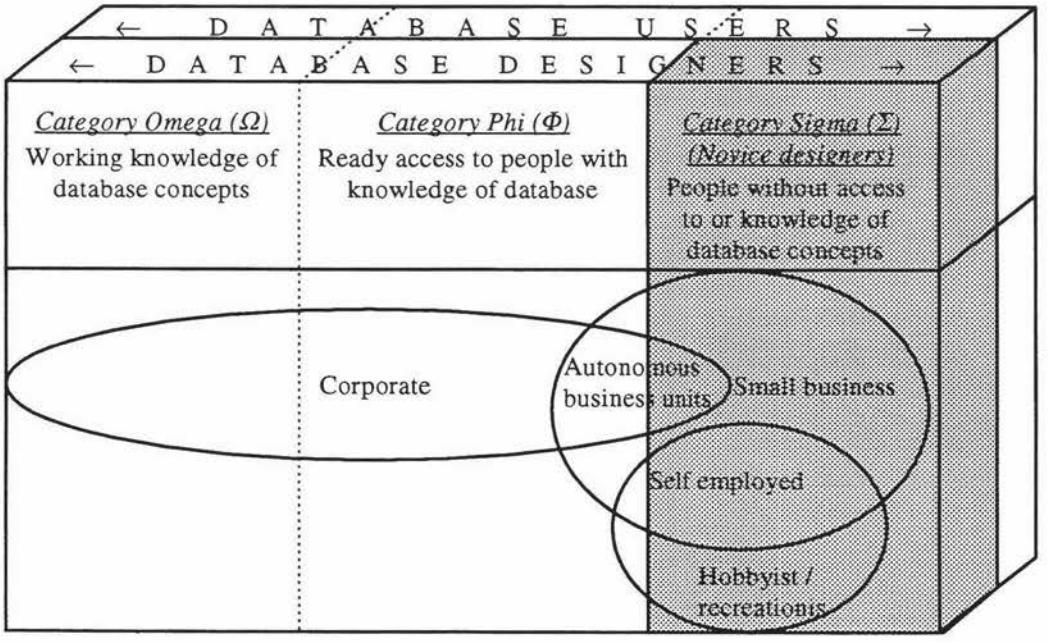


Figure 1. Classification of database usage environments.

The three vertical slices in Figure 1 represent different levels of understanding of relational database concepts. The first slice, termed category Ω, encompasses those that have a solid knowledge of relational concepts and issues concerning relational database design. These people might include, among others, information systems professionals, computer scientists, consultants and academics. The second slice, termed category Φ, incorporates those that do not possess a sufficient knowledge of relational concepts to enable them to design high quality relational database schemata,

but are able to acquire or utilise these concepts by virtue of having access to people from category Ω. This category includes, among others, managers and professionals within organisations that have some sort of information systems department. The third slice, termed category Σ and referred to as novice designers, contains people who are unfamiliar with relational database concepts and do not have any readily available means to acquire those concepts, yet wish to make use of relational database technology within their domain. These people have been described as autonomous end-users (Srinivasan, 1992) and may be found in small or single person organisations, autonomous business units or hobbyist and recreational areas that are isolated from any significant database education support structure. This set of locations is similar to the SOHO environment described earlier but includes the added class of autonomous business units. In the rest of the present report the term SOHO+ will be used in preference to SOHO where the subject of the text is an environment including autonomous business units.

1.1 DBMS Utilisation Environments

Five organisational working environments that are, or might in the future be, exploiters of database technology are also depicted in Figure 1 by three overlapping ellipses. These organisational working environments all require people to carry out both the user and designer functions to be able to make effective use of their database technology. However, while in categories Φ and Ω the design and end-use functions are likely to be performed by different personnel, in category Σ they are more likely to fall to the responsibility of a single individual. It is this dual function of the individual in category Σ that is the main focus of this report.

Traditionally relational database management systems, because of high cost and processing power requirements, have been products used almost exclusively in larger organisations. Recent commercial releases of low price, high performance relational database management systems have brought these products within the budget of

individuals and small organisations. Products such as Microsoft® Access®[1] and Borland Paradox[2] have extensive database capabilities, a reputedly user-friendly interface and a low purchase price making them good products for marketing to the small business sector and Σ database designer. Although these products do represent a scaling of the budget constraint barrier there is still another barrier, the conceptual barrier, preventing the effective use of these systems by Σ designers. The conceptual barrier is the exposure, understanding and familiarity with relational concepts that Σ designers do not have. It does not necessarily follow that because these products are affordable and readily available people are going to be able to use them productively.

Date stated "if you are a database professional, and don't know the relational model thoroughly, you are practicing medicine without a licence" (Date, 1993). If we accept that the intent of this statement is to say that solutions to tasks involving relational database concepts are likely to be of higher quality when undertaken by people with a sound knowledge of the relational model then the inverse should apply, namely that solutions undertaken by people who do not have an understanding of the relational model are likely to be of poorer quality.

It is likely that as relational database management systems permeate throughout the SOHO+ market there will be an increasing number of database designers that have little or no appreciation of relational concepts and therefore fall into the Σ category. In many of these instances the purchaser of the RDBMS will be its sole user and by default also the database designer. While it is possible to use a relational database management system without understanding relational concepts, designing a relational schema without this understanding is likely to produce a poor design that will result in problems when subsequently querying and retrieving data.

---

[1] Access® is a registered trademark of Microsoft®.

[2] Paradox™ is a registered trademark of Borland.

We can conceive then, of a distance between what relational database management system (RDBMS) vendors are supplying and what people in the targeted market can use effectively. Vendors have attempted to bridge this distance by making their products easier to use. Microsoft Access and Borland Paradox are two popular Microsoft Windows[3] based relational database management systems that exemplify an attempt to bridge this gap. These vendors have used common features offered by the Windows operating system and environment along with some innovative ideas to assist the novice designer to be able to utilise their RDBMS. However most of these features concentrate on the input, manipulation and reporting functions of the RDBMS. There is little assistance given to the question of how to produce schema designs. Unfortunately for someone wanting to get their database up and running, schema design is the first thing that must be done before they can put their RDBMS to use.

The assistance given by the vendors relating to schema design typically consists of a section in the instruction manual explaining what tables are, a sample database consisting of three or four tables and helpful innovations such as Access' table wizards that build tables for typical business or personal applications. While these facilities will help the designer create tables they do little to help the designer decide what tables to create for their own application area. For the purpose of creating a schema design the purchasers of these systems are mostly left to their own devices. The lack of vendor guidance for schema design, especially if compared with the assistance level offered for other database functions, may lead the novice into thinking that issues associated with this area are not particularly important. When this is combined with the selling emphasis placed on RDBMS flexibility, a novice is likely to think that one schema design should be as good as any other. Consequently, as supported later in the present report, the novice designer is likely to either seek outside assistance to design the schema or proceed through the schema design phase iteratively, building tables intuitively until a schema emerges that appears to satisfy the requirements of the area being addressed.

---

[3] Windows™ is a registered trademark of Microsoft®.

Setting aside the myriad of possible issues encountered during an intuitive design process, such as data redundancy, data duplication, normalisation, data inconsistency, storing derived data and others, a sufficiently persistent novice designer might design a database that will address the problem for which it was bought. An advantage of a relational database management system is its ability to extend beyond addressing the immediate and most obvious problems and support unanticipated questions and other business issues (Ricardo, 1990). However the extent of a database's ability to answer unanticipated questions can be considered to be less effective if the database design has been done in an intuitive, haphazard manner rather than using some more formal database design methodology (Date, 1993).

This report assumes, based on a self selected survey described in the main text and readings in the popular press (eg. InfoTech weekly, PC World, Computerworld), that there are many database applications designed by intuition and iteration being built or being used today in small, single person or autonomous business units. Some of these may have been in use for some time and might have large amounts of data contained within them. However, because few environments are stable over periods of time the database designer must eventually reflect changes that occur within the environment that a database models into the database structure itself. These changes can be of two types: 1) changes in data values; and 2) changes in the data structure. Changes in data values are the bread and butter of RDBMSs and should be handled relatively easily by the designer on a regular basis. Changes in the data structure (eg. an academic year changing from three terms to two semesters) are a less common occurrence and can have far reaching implications for the integrity of the database.

The designer, faced with the prospect of implementing a significant change in the data structure, can easily compromise the database's integrity (Ricardo, 1990). Having implemented a change the designer has no way of knowing whether an integrity compromise exists or not until problems begin to manifest themselves at a later time. By the time any problems caused by data structure changes are brought to the designer's attention rectifying them effectively may be beyond their capabilities. Cumulative complexities caused by likely additional data structure changes in intervening periods will make it unlikely that a designer could confidently trace back through prior changes and locate the cause of the problem, and even if they could it is

unlikely that the designer would know whether a different cure will prove to be of better quality than the first attempt.

The situation described here is not a worst case scenario, rather it is closer to a best case. In practice the novice designer is likely to build a database that, due to design flaws stemming from a lack of appreciation of good design practice, is cumbersome for initial use and difficult or impossible to alter for reliable future use (Batra et al, 1990). When novice designers find themselves with a database such as this they would appear to have three choices: firstly to leave their schema as it is and continue to use the limited functionality available; secondly to abandon the database; or thirdly to re-engineer the existing schema.

The first option, leaving well alone, is less than ideal for most situations as few data application areas remain stable over time. Leaving the database alone will see it become progressively less and less relevant. Consequently an organisation selecting this option will find its competitive situation gradually decreasing and eventually unable to support its information needs effectively.

The second option is a throw away option where all investment in the current database is abandoned. This option can come with a weighty price tag, the loss of the existing data resource that can represent a considerable investment. The attractive side to this option is the opportunity of being able to start afresh with a clean slate. If this option is taken the database sponsor (the organisation or individual for whose interests the database is being developed) should be aware of the reasons why the abandonment of the previous database occurred and take steps to ensure that these events do not recur.

The third option, re-engineering the existing schema, is potentially the most productive for the database designer and at the same time the most difficult. It is productive because it provides a strategy for the organisation to move from its current information platform to an information platform that is better matched to its current and foreseeable future requirements without abandoning its base of data. The difficulties arise in effecting the transition from the old schema to the new. It may be possible for the original designer to perform this transition if the database consists of a minimal number of tables. However as the number of tables increases so does the complexity

of the interrelationships between the data. This complexity will soon reach a point where the novice designer is unable to conceptualise the entirety of the database and therefore unable to effect a satisfactory transition from old to new schema. Another option available in this situation is to take the old schema and new requirements to a database professional. This places the onus on the professional to manage the transition. Unfortunately the professional is likely to have little or no appreciation of the existing schema nor the database's universe of discourse and will be dependent on the original designer to communicate both. Explaining the universe of discourse can be difficult and time consuming for the Σ designer but describing the schema, unless it is simple, is likely to be beyond their capabilities. It is this option and issues concerning this area that comprise the focus of this thesis.

This thesis proposes a method to evolve an existing schema through a migration procedure into a refined design. The eventual intended recipient of this procedure is the sigma designer. However the procedure, in its current state, is too complex for sigma designers and therefore requires a database professional to implement it. The procedure is suitable for implementation within an automated software tool at a future date. This software tool should hide sufficient complexity from the user to enable sigma designers to use it effectively.

2 The Research Programme

This chapter discusses the methodology underlying the research in the present report. The research incorporates a survey of the literature surrounding the subject, a hypothesis about database design, the classification of people who design databases and proposes a procedure to improve the quality of those designs. It begins with a survey of database designers that is followed by case studies of a selection of survey respondents. Finally it presents an experiment using one of the case studies to show the results that can be expected from applying the schema re-design procedure.

The first issue to be addressed is to define the parameters of the research. To do this it is necessary to formalise a research question. After surveying the relevant literature concerning this area of research the research question that arose is phrased: how can poorly designed database schemata be improved by people who are not familiar with database concepts.

A classification scheme is proposed in this report's introduction that places database users into different sections within a cube, depending on their level of involvement with DBMSs and their knowledge of database concepts. Using this classification scheme as a working tool the research restricts its main emphasis to one identified class, the novice designer. The validity of the classification scheme is a question that the report does not seek to prove but rather uses it primarily as a means to identify and focus on issues affecting a particular class of person. The findings in the report do however support the classification scheme to some extent, in particular the characteristics pertaining to the class of novice designer.

Having identified the characteristics of the type of person to concentrate on it is necessary to describe how they might be affected by issues involved in database design and implementation. To do this it is desirable to explore the literature on these subjects and extract commentary surrounding the subjects of database design and database usage. It should be noted here that the scope of the research and therefore

the literature review is limited to relational database as this is likely to be the preferred paradigm of novice designers. The literature review also devotes considerable space to aspects of the NIAM methodology as it is a derivation of this methodology that is proposed for improving existing database designs.

The research moves from the literature review into a survey phase where the objective is to identify the existence of novice designers and provide evidence of the type and nature of database issues that affect them. A further objective of the survey is to identify a small number of novice designers to participate as case study subjects. The survey phase of the research confronts an interesting problem; that of locating a sample population of novice designers. Due to the characteristics of this class of person they are likely to be thinly distributed throughout many areas of the population and in many industry sectors. Rather than strive for a significant random sample the research instead uses a combination of a self selected sample and a directed mail-out. This expedient method means that the reliability of inferences arrived at from statistical analyses of the survey results is reduced.

The next phase in the research is to examine in more detail some of the issues confronting novice designers. To accomplish this the research utilises four survey respondents as case study subjects. The purpose behind this case study analysis is to support assertions made in the early stages of the research about the kinds of database issues facing novice designers. In particular attention is paid to the design problems they have in common and the strategies they have used to build their databases and overcome these design problems.

The final phase of the research is described here as an experiment although it incorporates aspects of other research methods such as action research and case study analysis. During this phase a database schema re-design procedure is proposed and applied to two database designs. The first database is one well known by the researcher and is primarily used as a vehicle to demonstrate the concepts and methods used in the procedure. Consequently the schema re-design procedure is only applied to limited parts of this database schema The second database design is taken from one of the case studies and the schema re-design procedure is applied to the complete schema. While the case study's database design is undergoing the schema re-design procedure a

second researcher, experienced in data modelling and database design, is independently designing a database schema for the same universe of discourse but without seeing the original schema. The purpose of this phase is to assess the quality of the design that emerges from the schema re-design procedure against both the original schema and the second researcher's schema. Additionally, increases or decreases in typical novice designer issues present in the new schema will be noted.

3 Scope of the Study

The scope of the present study includes strategies concerning the effective implementation of changes to the schema of a poorly or haphazardly designed relational database. As a database professional, the author has had a number of poorly designed databases referred to him over the years. In most cases no-one knows much about the design of the database schema. What they do know, however is that it is unreliable and does not support all the functions that they want. A brief description of one of these cases, a Forestry database, helps to highlight potential problems arising from a widespread implementation of database systems in SOHO+ environments and focus the direction of the research.

A medium scale timber company used a relational DBMS to implement a system to track felled timber from the forest to the eventual buyer. They employed a contractor to build the system and were happy with it for some time. A while later business practices changed and the company needed to reflect these changes in its timber tracking system. Unfortunately the creators of the system were unavailable to perform the updates so the company sought the researcher's advice. With the researcher knowing little about the timber business and the company employee knowing little about database concepts it proved to be a difficult and time consuming process to critically examine the schema design. In this case little progress was made toward solving the company's problems other than to make an assessment of the schema design as being inherently of poor quality. Much of the data was duplicated across different tables, table key structures were difficult to comprehend and several tables seemed not to relate to the rest of the database at all. The point that this case highlights is the intractable problems designers can face from an existing, poorly designed database. The remedial options open to a novice designer in this predicament are limited and it is the issues concerning these remedial options that are relevant to the present report.

The research draws heavily on issues concerning relational database design that underpin this focus. These issues include relational theory, conceptual data modelling, logical data modelling, physical data modelling, relevant aspects of computer assisted software engineering (CASE), reverse engineering of data, appropriate levels of normalisation, the relationship of functional analysis to database design and the management of derived data.

Although there are many types of databases being used productively in the SOHO+ environment, such as flat-file, relational, extended relational, object-oriented and others, this study is primarily interested in DBMSs that support relational principles as specified in Codd's seminal paper (Codd, 1970) because it is these relational based products that represent the main thrust of DBMSs being marketed to the SOHO+ environment. A question, addressed in the body of this report, that arises when using this scoping parameter is how to classify what is and what is not a relational database management system. If the term relational is used in a tight sense then this will be a set containing no commercial products as none currently available support the full set of features of Codd's original model. On the other hand if we accept the use of the term in a loose sense it can then describe almost every database system available (Mattison, 1993).

Discussion of conceptual data modelling is highly relevant to database design. Indeed the conceptual model is a powerful tool used to conceptualise and communicate a universe of discourse. It is likely that many of the problems associated with database schemata designed by novice designers stem from an underestimation of the importance of gaining a holistic conceptualisation of the universe of discourse. There can be a tendency to slip into design myopia if the emphasis is wholly on the immediate problem (Mattison, 1993). Conceptual data modelling is a tool available to both novice and professional. This report deliberates on the differing considerations between designing the database schema and facilitating the functional requirements of the universe of discourse. This topic is relevant because novice designers are more likely to emphasise the immediate functional requirements than consider long term needs.

Data modelling is a technique also used when moving from a conceptual design to a design that is directed toward a particular type of database management system. This activity is termed logical data modelling and in a relational sense is a depiction of the core data entities within a universe of discourse and the interrelationships between those entities (Hull and King, 1987). The logical data model ignores physical design considerations and therefore can be considered as a purist's view of the data. There is a number of logical data modelling techniques proposed by theorists and practitioners that will be discussed in relation to their applicability to the issues encountered in re-engineering database schemata.

Physical data modelling is the last activity taken before building the physical database schema. It produces a depiction of how the universe of discourse is to be implemented into a particular database management system. A physical relational data model is usually depicted using similar constructs to those used when producing the logical model, but it differs by presupposing that a particular database product has been chosen. The physical relational data model shows any compromises and design considerations made to suit the particular mechanisms of the selected DBMS. It also shows compromises included to better tune the system taking into account the functional and performance requirements of the implementation environment.

The needs of database designers tackling problematical, messy design situations have not been ignored by systems developers. Computer assisted software engineering (CASE) tools offer computerised assistance throughout the system's development life-cycle to the developer. A relatively small facet of CASE addresses issues relevant to database design. This facet offers tools and techniques that can benefit the database designer. CASE tools tend to primarily support one particular systems analysis and design methodology (Ryder, 1993). The appropriateness of particular methodologies to database re-engineering issues are discussed in the body of this report. Many CASE tools are developed with the needs of large organisations in mind and consequently are priced well beyond the budgets of designers in the SOHO+ environment. There is a slowly filling gap in the CASE tool market for tools designed to address problems encountered by designers in the SOHO+ environment. This report briefly discusses a few of the products (Meta-case, S-Designer, IBMS, HAT) filling this gap.

3.1 Reverse Engineering of Database Schemata

Reverse engineering is the term associated in the software engineering literature (Premerlani et al, 1994) with the analysis and understanding of existing systems, and the ability to record the results of this analysis in an accessible manner. This is often done by creating models of the existing system in much the same way as traditional systems analysis creates models of non-existent systems (Sabanis and Darlison, 1992). This report narrows the scope of reverse engineering to address the reverse engineering of database schema and data rather than systems, the emphasis being on using the concept to produce a logical and conceptual data model from a combination of the knowledge held by people familiar with the universe of discourse and the physical files that make up the current database schema.

Schemata arrived at through design by trial and error are likely to contain different levels of normalisation ranging from non-first normal form (NF^2) to second or perhaps third normal form. This research addresses issues concerning desired levels of normalisation for each stage of the proposed reverse engineering specification. Each of the conceptual, logical and physical data models serves different purposes and is therefore suited to different levels of normalisation, what may be appropriate for one type of model is not necessarily best suited to another.

3.2 Data Modelling

Data modelling can, for those practicing it, become an end in itself. It is possible, and sometimes desirable, to strive for a pure data model, i.e., a highly normalised data model that captures the complete universe of discourse. However, often this view of the data structures is not appropriate for implementation due to possible mismatches with the functional requirements of the proposed system. There is a school of thought that advocates data modelling and database design be carried out in conjunction with functional analysis (Batini et al, 1992). This pragmatic approach to systems development is likely to appeal to database designers in the SOHO+ environment where the emphasis is often on getting the immediate job done and getting it done as quickly as possible.

One issue that impacts upon the design of database schemata concerns the modelling, processing and storing of derived data. It can be difficult to distinguish between base data and data derived from the base data when the designer is either unfamiliar or too involved with the universe of discourse. Data modelling at the conceptual and logical levels done thoroughly requires that this issue be teased out and resolved so that the model contains no derived data. However at the physical level it is likely and often desirable that derived data be stored for processing purposes depending on the focus within the universe of discourse. This report explores strategies for identifying derived data within an existing schema and translating that data into its base data at the conceptual level.

While the focus of the research is centred on specifying a solution for problems encountered by novice designers in the SOHO+ environment it is not confined to these parameters. Indeed the solutions are equally applicable to any database designer who neither has access to a CASE tool that will assist in their database design nor the expertise and conceptual understanding to use a substantial database design methodology.

SECTION TWO

RESEARCH CONTEXT

4 Context

This chapter provides a context for the research issue particularly in the areas of relational database design and RDBMS adoption by different sectors of society. It begins with the history and evolution of relational theory paying particular attention to schemata design issues. The report then addresses data modelling, how it dovetails with relation database design and how relevant and useful it may be to novice designers. In particular it discusses the NIAM methodology, the entity relationship diagram (ERD), extended entity relationship diagrams (EER), and semantic data models and comments on how these relate to the different emphases of the conceptual, physical and logical levels of data modelling. Rules and heuristics for mapping between these levels of data model are then discussed. Different analysis and design methodologies are introduced into the report and investigated with regard to the level of importance they place on relational database design. Formal relational schema design conventions such as normalisation and the management of derived data are then discussed in conjunction with their relevance for novice designers. Finally possible analogies to the adoption process of relational database management systems by novice designers, such as the adoption of spreadsheet technology by managers, are described with a view to investigating whether strategies used to resolve issues successfully in that area can be applied to the novice designers' adoption of RDBMSs.

4.1 The Position of the Relational Model

Current commercially available database management systems are generally modelled using one of three widely used classical paradigms developed during the 1960s and 1970s: hierarchical (Tsichritzis and Lochovsky, 1976), network (CODASYL, 1971) and relational (Codd, 1970).

4.1.1 Hierarchical Model

The hierarchical database paradigm is the first database paradigm to have established wide acceptance in commercial environments. Hierarchical database management systems are dedicated to the capture and control of a specialised data relationship called a hierarchy (Mattison, 1993). This hierarchy is often referred to as a tree structure containing many nodes, of which the node at the highest level is called the root. The root node can have many subordinate or child nodes but does not have a parent. Every other node in the hierarchy can also have many child nodes but must have one and only one parent node (Ricardo, 1990). In relationship terms this equates to a one-to-many relationship between the parent and its child nodes, the inverse relationship being one-to-one relationship between each child node and its parent. This is exemplified in Figure 2 that shows a hierarchical data structure for cars sold in a car sales yard. Here the customer segment represents the root node and has two child nodes one of which has a further two child nodes. Each hierarchical DBMS has a set of navigation commands that allows the user or program to move up, down, left or right through the structure.

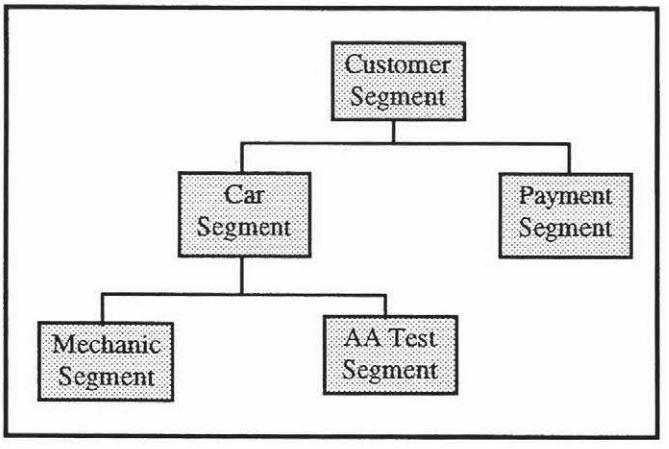


Figure 2. Hierarchy diagram for a car yard

Hierarchical DBMSs have built up a long and productive reputation in large organisations over time and consequently retain a substantial installed base in organisations using mainframe architectures. They offer a number of features that are likely to endear them to large organisations. First they offer fast and efficient performance for a universe of discourse where the operational requirements are stable

and predictable. Second, the logical data relationships mapped in to the database are relatively easy to understand. Third, they can be tuned to run at an optimum level of efficiency and then left to function with minimal maintenance (Mattison, 1993).

4.1.2 Network Model

The second database paradigm to achieve wide commercial acceptance and provide a viable alternative to hierarchical DBMSs is the network database paradigm. This paradigm is primarily composed of three types of data constructs: data items (fields), record types (entities) and sets (logical relationships) (Mattison, 1993). The data item is associated with a data type and is the network DBMS's smallest named unit of data. A collection of data items makes up a record, a quantity of which makes up a record type. The network DBMS set construct expresses relationships between record types (Ricardo, 1990).

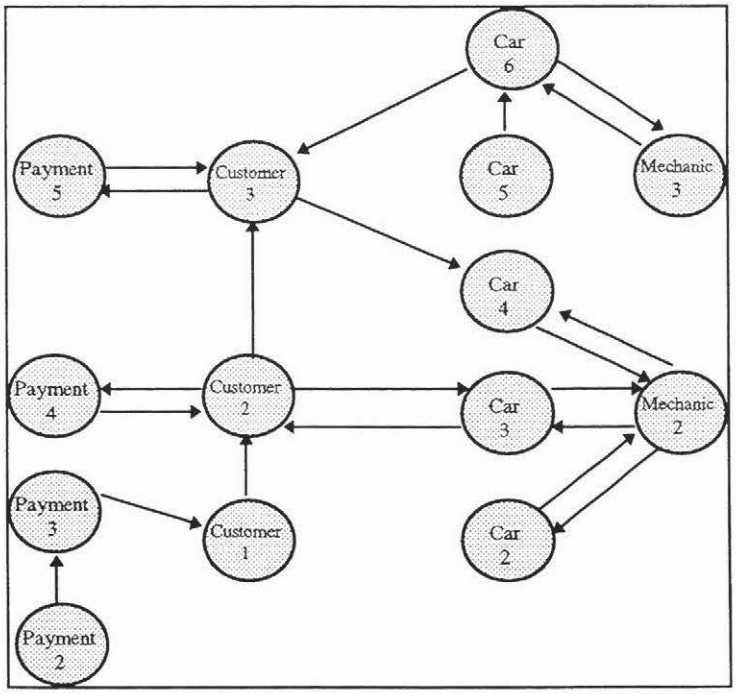


Figure 3. A network diagram (adapted from Mattison, 1993).

The network database paradigm differs from the hierarchical database paradigm in terms of its allowable relationships. The one-to-many relationship from parent to child can be reversed in the network paradigm allowing a child node to have many parents -- or even be its own parent. Network structures are often diagrammed using bubble

charts. The chart in Figure 3 (adapted from Mattison, 1993), shows the relationships between Customer, Car, Mechanic and Payment in a car sales database.

Although more complex, the network database paradigm still offers fast and efficient performance for many applications and is still chosen to support a significant number of new mainframe systems being developed (Mattison, 1993). Network database systems are however not suitable for the XSOHO environment as they require a highly skilled team to build, manipulate and maintain the database system.

4.1.3 Relational Model

The third commercially accepted classical database paradigm is the relational paradigm. The present report focuses on this paradigm because it is through commercial DBMSs developed utilising some of the precepts of the Relational Model that RDBMSs are permeating into a wider section of society. The relational database paradigm has, at the expense of the hierarchical and network paradigms, increased its popularity over time due in part to features such as simplicity, facilitation of data independence, mathematical rigour and an ability to support non-procedural database query languages (Srinivasan, 1992).

The Relational Model was first proposed in 1970 in a paper written by Codd called "A Relational Model of Data for Large Shared Data Banks" (Codd, 1970). Although there are a few papers published earlier dealing with aspects of the Relational Model, eg. Codd (1969), this paper is widely regarded as the seminal article concerning relational database theory (Date, 1986a). The Relational Model consists of three major parts: a structural part, an integrity part and a manipulative part. The structural part consists essentially of n-ary relations together with their underlying domains. The integrity part consists of two general integrity rules, namely entity integrity and referential integrity. Finally, the manipulative part provides a set of algebraic operators or their calculus equivalents for data manipulation in all of its manifestations. (Date, 1986, p.314). The Relational Model offered improvements for database management by:

> *"protecting users of formatted data systems from the potentially disruptive changes in data representation caused by growth in the data bank and changes in traffic" (Codd, 1970, p.387).*

Between 1970 and 1975 Codd continued to publish papers that further defined and refined the Relational Model (Codd, 1971a,b, 1972a,b, 1974a,b). During this time the first prototypes and commercial RDBMSs became available ((Czarnik et al., 1975); (Todd, 1975); (Zloof, 1975); (Astrahan, 1976), (Stonebraker et al., 1976)). While these products were based on precepts of the Relational Model, they failed to incorporate many of the features identified as components of the Relational Model in Codd's seminal paper (Tsichritzis and Lochovsky, 1982).

Codd bases the Relational Model on elementary relation theory and applies this theory to the problems of data dependence and data inconsistency that affected systems prior to 1970. Data dependence occurs when some data representation characteristics are changed, but performing those changes results in an impairment of the performance of some application programs dependent on that data. Dependence of this type occurs in three principle forms: ordering dependence, where the particular storage ordering of the data conveys some additional meaning that will be lost or altered if the storage ordering is changed; indexing dependence, where data indexes cannot be removed, changed or updated without adversely affecting the performance of application programs dependent on that data; and access path dependence, where application programs are adversely affected if data structures such as tree-structured files are changed in a formatted data system (Codd, 1970).

Data inconsistency, while not such a serious problem at the time of Codd's article, was likely to become so as the size of data banks increased over time. Most inconsistencies stemmed from the storage of redundant data required to explicitly link existing data, and ensuring that each instance of duplicated data had the same value and that that value was the current value (Codd, 1990).

In pre-relational times, as a result of computer systems becoming more powerful they acquired the potential to manage larger and more diverse databases. The increasing size of databases not only meant putting more data records into the database but often increasing the complexity of the inter-relationships between the data records.

Consequently as the complexity of the stored data structures increased so did the level of system overhead required to maintain data consistency (Codd, 1990).

Performing queries on the data in the database becomes an increasingly complex task as the complexity of the stored data structures increases. When data structures are relatively simple it is possible for database administration personnel to anticipate a set of likely queries and write code to perform those queries efficiently. However, as the scope of the stored data increases it becomes more difficult to anticipate a reasonably comprehensive set of likely queries. Consequently the database administration personnel are faced with trying to keep up with the demand of coding query routines for a wider group of people wanting answers to a more diverse range of questions. In order to break the cycle of writing more and more code to answer more and more questions it becomes desirable to use a DBMS with an efficient query interface that enables quick and efficient response to ad-hoc queries.

The Relational Model addresses the issues highlighted above by providing a means to describe data with its logical structure only, the relational view of data, without superimposing any additional structure for machine representation purposes. This self descriptive structure leads logically to constructs and concepts that are central to the Relational Model (Date, 1986b).

4.1.4 Relations

The relation construct as contained in the Relational Model is borrowed from the field of mathematics and referred to in its mathematical sense throughout Codd's (1970) seminal article. It is generally represented conceptually as an array, as shown in Figure 4, although this representation may have little resemblance to its actual form. In a relation, rows are called tuples and columns are called attributes with the number of attributes defining the degree of that relation. In Figure 4 the relation *STUDENT* has a degree of four.

STUDENT

Student ID	*Surname*	*First Name*	*D.O.B*
65548751	Smith	Jill	23/08/46
66547899	Jones	David	21/05/48
65896547	Davis	Robert	24/06/46
67581259	Green	Jenny	05/08/39
64589654	Peters	Mary	01/05/47

Figure 4. Student relation

Data values contained in the columns are given meaning by their column names and can be restricted to members of a defined set through the use of the domain concept. Domains are a defined set of legal values that an instance of a particular column can assume (Date, 1986b). Domains can have a finite or infinite set of values, for example the domain of "planets in the solar system" is finite, while the domain of "positive integers" is infinite.

The generally accepted definition of a relation is as follows (Date, 1986a):

> *"Given the sets S1, S2,. . ., Sn (not necessarily distinct), R is a relation on these n sets if it is a set of n-tuples, the first component of which is drawn from S1, the second from S2, and so on."*

The present thesis will follow Date (1986b) in referring to a relation and a two-dimensional table as the same thing with the following properties:

- Each row represents an n-tuple of the relation,
- The ordering of the rows is immaterial,
- All rows are distinct,
- The ordering of columns is not significant,
- The semantics of each column is conveyed by its label and its domain.

Codd proposed that the user should interact with a relational model of the data consisting solely of a set of relations. "Each user should need not know more about any relationship [relation] than its name together with the names of its domains [columns]" (Codd, 1970. p.380).

In order to be able to perform many of the required functions on a set of relations there must be a means of joining them. Joins can be made using any two attributes from the same domain, however to ensure integrity the join operation is performed after defining

keys for each relation. Because each row in a relation is distinct there must be an attribute or set of attributes that uniquely identifies each tuple, this is a primary key. If the primary key of one relation is included in another relation, then in the second relation this is known as a foreign key (Date, 1986b). A join is performed by associating an attribute(s) of one relation with an attribute(s) from the same domain in another relation using a query statement.

4.1.5 Normalisation

Given that a relational data model consists of a set of relations there needs to be a way to organise these relations and decide what attributes each should have, this is something that happens during the process of normalisation (Codd, 1971a, 1972a). To rigorously normalise relations it is imperative that the designer understands the semantics of the data represented and is familiar with the data's universe of discourse. It is not possible to reliably derive the semantic nature of the data from a sample of tuples, as any set of tuples is unlikely to contain all possible data permutations. Novice designers, by definition, are unlikely to be familiar with normalisation concepts, therefore, although they will probably be familiar with the universe of discourse and the semantics of the data, they are not going to be in a position to organise their relations and attributes using the process of normalisation. A database structure designed with no appreciation of normalisation is likely to consist of relations in 0^{th}, 1^{st} or 2^{nd} normal forms, relations in these lower levels of normalisation result in a significant risk of producing inconsistent results to queries and are usually less amenable to changes imposed by the universe of discourse or changing functional requirements over time (Codd, 1990).

There are seven established levels of normal form illustrated as a series of nested circles by Ricardo (1990) in Figure 5. Higher levels have been proposed that put normalised data in a purer form (Fagin, 1979). Each level of normal form must satisfy the requirements of the preceding level of normalisation in order to be able to progress a step further towards a "better" design. Of the first seven normal forms the first three were defined by Codd (1972a), the fourth by Boyce and Codd (Codd, 1974b), and the final three at a later date by Fagin (1977, 1979) and Zaniolo (1986).

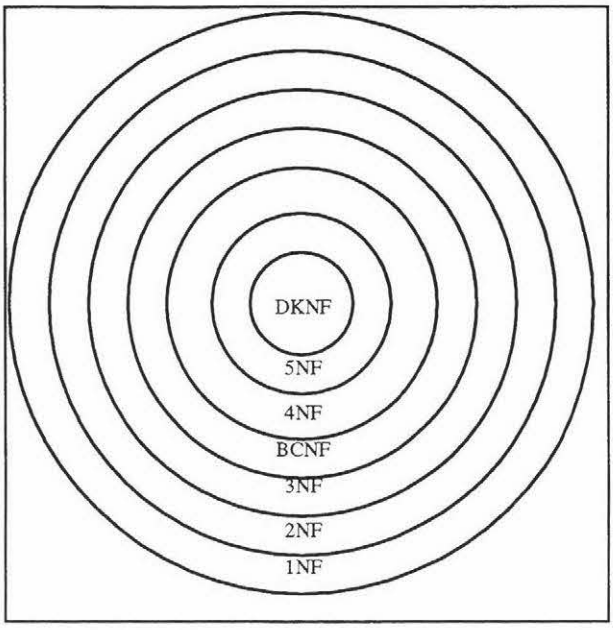


Figure 5. Normal forms (Ricardo, 1990).

Putting relations into higher normal forms is usually done to reduce the propensity for insertion, update, and deletion anomalies and to reduce the amount of redundant data in the database. The intention of normalisation is to maintain the integrity of the data and to make the meaning of the data clear and therefore easily understood. The process of moving relations into higher levels of normalisation usually involves the decomposition of existing relations into relations of smaller degree. While this process gives a "better" representation of the data it tends to come at the price of reduced physical performance because the increase in the number of joins required for many operations impacts adversely on processing time. Consequently for implementation purposes there is often a trade off between a pure design and a design that gives acceptable physical performance, this means that some relations in the database structure are likely to be in a different normal form to others.

A more thorough discussion of normal forms and normalisation is beyond the scope of this report and can be found in (Kent, 1983) and (Date, 1986b).

4.1.6 Sub-language

According to Codd (1970) the adoption of a relational model of data permits the development of a universal data sub-language. This sub-language should be descriptive in nature, rather than procedural, and capable of describing, querying or

manipulating any part of the database. Relational algebra and relational calculus were developed in the early 1970s as languages to manage relations. Although both languages proved to have equal expressive power Codd (1970) indicated that relational calculus would be easier to use and more appropriate as an interface to a database management system.

By the early 1980s there were several domain calculus languages including FQL (Pirotte & Wodon, 1977), DEDUCE (Chang, 1976), QBE (Zloof, 1975) and SQL. The American National Standards Institute (ANSI) adopted their own version of SQL (Structured Query Language) as the standard sub-language for relational database management (X3H2, 1985). Since then there have been criticisms of SQL both in terms of its ability to fulfil all of the requirements of a sub-language within the Relational Model and the number of different versions of SQL that are used in relational database environments (Date, 1986a).

Other languages such as SEQUEL (Structured English Query Language) (Chamberlin and Boyce, 1974) and QUEL (Query Language) (Stonebraker et al, 1976) arguably support as much, if not more, of the Relational Model as SQL but because of the adoption of SQL by ANSI, the use of QUEL and SEQUEL is limited to few products.

Many RDBMSs marketed for use in the SOHO+ environment offer substitutes or alternatives to sub-languages such as SQL. Products such as DBase[4], Access and Paradox provide 'query by example' interfaces for querying the database and menu options with 'fill in the form' templates to define tables, attributes and other RDBMS objects rather than a textual data description language. To a large extent this obviates the need for novice designers to learn a sub-language such as SQL, although an interesting change to recent releases of Paradox and Access is the inclusion of an SQL translator that converts QBE queries into SQL syntax. This translator however does not at present translate data description actions into the equivalent SQL syntax.

---

[4] DBase™ is a registered trademark of Borland.

4.1.7 Operations on relations.

The Relational Model stipulates that there be a set of algebraic operators, or their calculus equivalent, for data manipulation in all its manifestations (Date, 1986b). These operators include union, intersection, difference, product, select, project, join and divide and are not just intended for data retrieval but for the construction of arbitrary relation-valued expressions. Using these operators gives rise to the opportunity for investigating numerous aspects of database systems including database design, query optimisation, view definition, database restructuring, and security and integrity (Codd, 1990).

The products used by novice designers, while incorporating data manipulation tools, do not cater for all types of manipulation. Functions commonly excluded from products such as Access and Paradox include view definition, query optimisation and verification of database design.

4.1.8 RM/T

During 1979 Codd presented a paper to the Australian Computer Society meeting in Tasmania defining an extended Relational Model called RM/T (Codd, 1979). RM/T defines 333 features of the updated Relational Model, an increase of 280 over the original (Codd, 1990 p.10). As the title of this paper, "Extending the Database Relational Model to Capture More Meaning", suggests, the objective of RM/T is to enhance the semantic capability of the original Relational Model.

During the 1980s Codd continued to publish papers in support of the Relational Model (Codd, 1982, 1986). In 1990 the content of all of these papers was brought together in a book "The Relational Model for Database Management version 2 (Codd, 1990). It is in this book that Codd acknowledges the lack of vendor support for RM/T and proposes a gradual release of RM/T's features. Each five years he proposes a new version of the Relational Model will be released allowing vendors and customers time to understand each version's additional features. The original is called RM/V1 and subsequent releases RM/V2, RM/V3, etc (Codd, 1990).

The period between Codd's publication of RM/T (Codd, 1979) and his stated desire to release a series of versions of the Relational Model each containing more RM/T features (Codd, 1990) has seen substantial comment on RM/T ((Date, 1986b), (Hammer & McLeod, 1981), (Balfour, 1988), (Smith and Smith, 1977)) but there have been few attempts to implement it in commercial products.

As mentioned earlier RM/T is a substantial update to the original Relational Model and provides an indication of what RDBMSs may incorporate in the future. The present report draws a few concepts from RM/T that are relevant to conceptual database design and data modelling but omits comment on the bulk of RM/T as it covers topics outside the scope of this thesis.

RM/T is in part a response to criticisms that the Relational Model does not capture enough of the semantics of the data ((Smith and Smith, 1977), (Hammer and McLeod, 1978)). Therefore among other concepts RM/T introduces sub-types and super-types of entities, as shown in Figure 6.

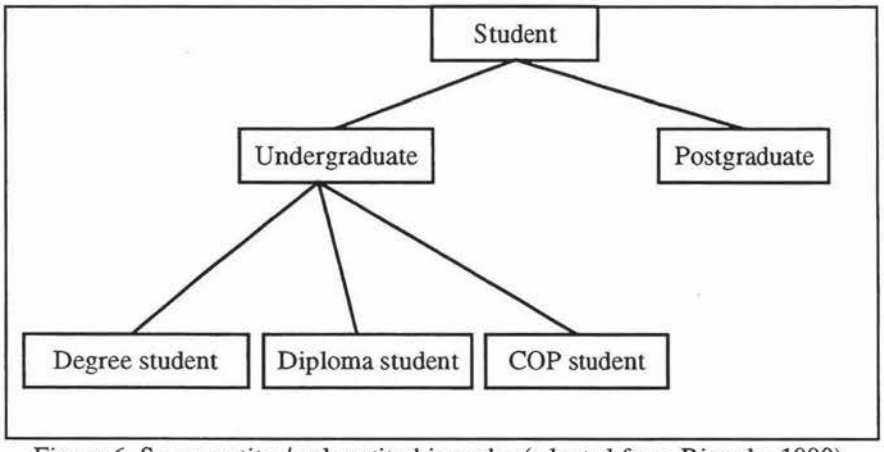


Figure 6. Super-entity / sub-entity hierarchy (adapted from Ricardo, 1990).

Sub-entities and super-entities are one logical implication of generalisation as defined by Codd (1979). This concept involves a hierarchy of relations where the relation at the top of the hierarchy carries attributes that apply to all the relations lower down in the hierarchy. Figure 6 gives an example of this concept where Student is a super-entity of Undergraduate, Degree student is a sub-entity of Undergraduate and Postgraduate is a sub-entity of Student (adapted from Ricardo, 1990).

4.2 Other Data Models

Databases prior to 1969 were predominantly based upon the hierarchical or network Data Models and designed without the assistance of any formalised data structuring techniques (Tsichritzis and Lochovsky, 1982). During 1969 Bachman published a paper entitled "Data Structure Diagrams" proposing a graphical depiction of the relationships between data in a particular universe of discourse (Bachman, 1969). The application of these diagrams to represent the physical design of network databases marks the beginning of the introduction of a number of new and innovative Data Models.

It is desirable at this point to be precise about the meaning of terms used in this section of the thesis.

- The term Relational Model (using capitalisation of the first letters) refers to the Relational Model of Data as Codd defined it (Codd, 1970).
- A relational data model (no capitalisation) refers to the conceptualisation of a logical or physical relational database schema (Loosley and Gane, 1989).
- A Data Model (capitalisation) refers to a formally described method of representing or modelling data. The Relational Model and Hierarchical Model are examples of Data Models.
- Data modelling is the process of mapping a universe of discourse to a particular Data Model.
- Data diagramming is a component of data modelling and refers to the method and conventions used to produce a graphical data representation of a universe of discourse.

Using the techniques and ideas contained within different Data Models people can apply alternative approaches to the analysis and design of the data in a universe of discourse. Whilst the trained database professional may have a comfortable cognitive fit with the constructs of the Relational Model, novice designers are likely to have a wide variety of mental models that may be more receptive to the constructs available in other Data Models.

4.2.1 The entity relationship model (ERD).

In 1976 Chen proposed the Entity-Relationship Model (ER) as a modelling technique to represent a view of an enterprise's data, independent of storage and efficiency considerations. The ER approach was primarily designed to define a stable enterprise schema that would map on to a network or hierarchical DBMS (Chen, 1976).

The popularity and wide-spread use of the ER approach in industry is arguably due largely to its associated diagramming technique (Date, 1986a). This diagramming technique consists of a formalised a set of data diagramming conventions that can be usefully employed as a documentation aid, often during the database design process. Its primary constructs are as follows:

- Each entity type is shown as a rectangular box, labelled with the name of the entity type in question.
- Each relationship type is shown as a diamond-shaped box, labelled with the name of the relationship type in question.
- Each relationship diamond is connected by lines to the entity boxes for the entities participating in the relationship in question.
- Each connecting line is labelled 'one' or 'many' to indicate whether the relationship is one-to-one one-to-many or many to many.

Codd, however, asserts that the ER Model is not a Data Model in the same sense of the word as the Relational Model rather it is little more than a diagramming technique (Codd, 1990). His main criticisms are its "lack of precise definitions, lack of a clear level of abstraction, and lack of mental discipline" (Codd, 1990 p.476). He focuses criticism on the relationship construct arguing that while one person may justifiably identify something as an entity another person may, also justifiably identify that thing as a relationship.

4.2.2 Extended entity relationship models (EER).

Extensions to Chen's original entity relationship model, known as extended entity relationship models, have been proposed by many authors (Teory, Yang & Fry, 1986) amongst others. Although each of these retains the ER model's characteristics of

representing information in terms of entities, their attributes and relationships, researchers have added different constructs that increase the model's ability to depict complexities more efficiently.

Concepts addressed by researchers include:

- The abstraction concept of generalisation ((Navathe & Cheng, 1983), (Elmasri et al., 1985), (Ling, 1985), (Atzeni, et al., 1981), (Scheuermann et al., 1980)).
- Ternary relationships and composite attributes (Ling, 1985).
- Existence constraints ((Webre, 1981), (Sakai, 1983)).
- General integrity constraints ((Lenzerini, et al., 1983), (Oren, 1985)).

Constructs found in EER models are often applicable to complex modelling situations. However before employing EER constructs the modeller must recognise the complexity involved in the universe of discourse. The likelihood of this recognition will usually be dependent upon previous modelling experience. It seems unlikely that tools afforded by EER are going to be utilised by novice designers as by definition they are likely to have neither the familiarity with modelling tools nor the modelling experience required to model the complexity contained within their universe of discourse.

4.2.3 Relational Model replacements

Over the years several alternative models have been proposed to replace the Relational Model. One of the rationales driving the publication of these alternatives is the perceived shortcomings of current RDBMSs. While it is fair to say that current RDBMSs have their shortcomings, protagonists regularly defend the Relational Model by pointing out that current RDBMSs support only a subset of the concepts described in the Relational Model. If the Relational Model were supported fully, Codd (1990) argues, then most, if not all, of the reasons for finding a Data Model to replace the Relational Model would cease to exist. Of the Relational Model replacements published thus far none have achieved significant support among database management

systems' vendors. A brief description of four of these proposed replacement approaches follows:

- The universal relation approach (UR).

  This approach was proposed in 1984 and states that a universal relation can replace the entire Relational Model (Maier, Ullman and Vardi, 1984). A universal relation is defined as being composed of one very large table containing all of the attributes of the database upon which the required functions can be performed. While UR is one extreme view of a relational database that the Relational Model should support, Codd (1990) claims that it is "preposterous" to believe that it can provide an effective total replacement.

  Of the eight described reasons supporting Codd's refutation of Vardi's claim (Codd, 1990) his strongest objection is based on the algebraic operators used to create the universal relation. The Relational Model has ten distinct kinds of theta-joins each based on one of ten comparators thus providing considerable flexibility. On the other hand the construction of the universal relation requires that one and only one of these theta-joins be used thus immediately restricting the power and flexibility offered by using the other nine theta-joins.

- The binary relation approach (BR).

  This approach first proposed in 1973 and later in 1988 (Mark, 1988) is based on the mathematical claim that any problem expressed in terms of relations with degree higher than two (ie. three or more attributes) can be reduced to an equivalent problem with relations of degree one or two. It is in a sense the opposite to the universal relation approach.

  Codd's (1990) objections to the binary relation approach centre around the additional volume and complexity incumbent with any but the simplest of relational schemata using this approach. As the complexity of a schema increases, so the overhead imposed by decomposing the schema into binary relations reduces the efficiency of the database. Supporting composite key structures poses particular problems for the binary relation approach because it

requires numerous additional tables. The expression and conception of integrity constraints also become more difficult due to the large number of tables required.

- Semantic data models.

  The objective of these approaches is to capture the meaning of the data without adding unnecessarily to its complexity (Hammer and McLeod, 1981). However, the level to which this objective can be achieved is a topic currently being debated in the literature (Codd, 1990).

- The object-oriented approaches (OO).

  These approaches attempt to put forward and define more abstract data types (Cattell, 1991). While this direction is a step forward for programming languages is unclear whether the same applies to database management systems (Codd, 1990).

  Object-oriented DBMSs are currently in their infancy. While some products such as Ontos[5] strongly embrace the Object-oriented paradigm many products claiming to be object-oriented could be more accurately describe as extended relational DBMSs, ie. relational with some object-oriented concepts added.

4.3 Relevance to novice designers

The Relational Model as described earlier has seen wide implementation in commercial DBMS products during the past twenty years particularly on PCs and hence it is the model that has been chosen for this research. Although many of the Data Models briefly described in this section contain the words 'data model' in their title most fall short of the completeness criteria required by Codd's definition of a Data Model (Codd, 1990).

---

[5] Ontos™ is a registered trademark of ONTOS Inc.

Relational database management systems are now readily available to the general population for application to non-trivial problems. To build effective systems, these people not only need user-friendly software, but a basic knowledge of techniques about systems analysis and design as well. Since most information systems have a database component, it is important that these people understand something of the database design process and techniques. If the people designing databases have no appreciation of the existence of data models and the relevance data models have to designing database schemata then it seems likely that their database schema design and consequently their systems will be of inferior quality. One option available to improve the quality of the schema, aside from starting again from scratch, is to invoke a schema redesign procedure. Schema redesign, for the purpose of this thesis is synonymous with reverse engineering of database schema.

4.3.1 Reverse engineering

The case for reverse engineering data is first presented in papers by ((Sneed and Jandrasics, 1988), (Choobineh et al., 1988), (Bachman, 1988), (McWilliams, 1988)) as a technique to improve the maintainability of existing systems. These papers discuss the broad view of reverse engineering as it applies to the complete systems development life cycle (Figure 7).

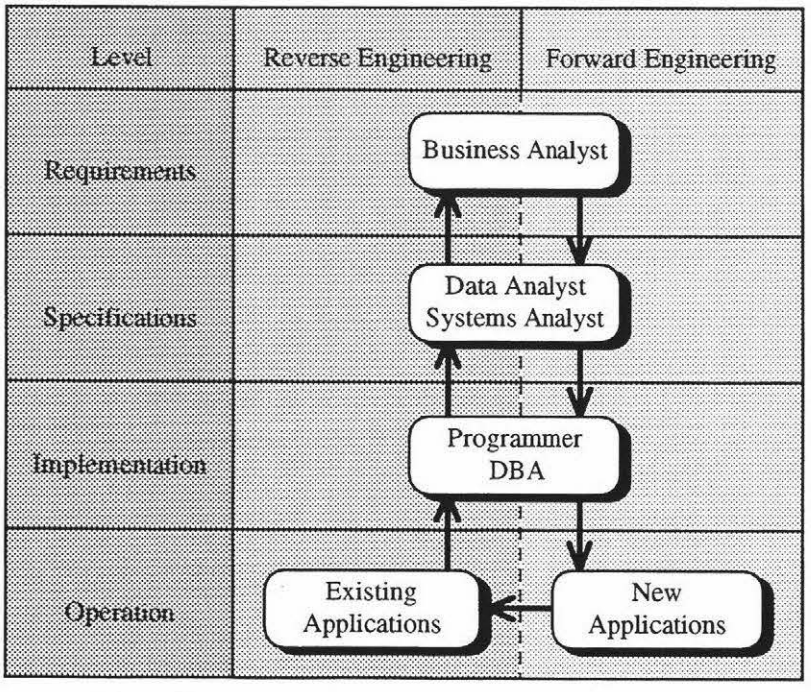


Figure 7. Re-Engineering Cycle (from Bachman, 1988).

Reverse engineering is discussed primarily as a component of CASE. Indeed Bachman (1988) sees the successful future of CASE inexorably linked to the development and inclusion of sophisticated reverse engineering capabilities within CASE tools.

> *"Originally CASE concepts and products focussed solely on creating new applications systems, but such applications represent a rapidly shrinking market ..... all that is needed for CASE to acquire a broad base of usage is the ability to revise the existing applications quickly ....."* (Bachman, 1988, p.49).

Aspects of CASE tools that are associated with the specification and maintenance of relational database schema are a relatively small part of integrated CASE tools currently available. Consequently the effort put into incorporating reverse engineering features for relational schemata has been fragmented by the concurrent effort to include reverse engineering features for other aspects of CASE such as program code and functional process specification (Sabanis and Darlison, 1992). Much of the focus in the area of reverse engineering data has centred on recovering the physical database design from existing schema and storing it in a repository (Rock-Evans and Hales, 1990).

As Sabanis and Darlison (1992) point out however it is at the point where there is an accurate representation of the current physical schema that analysts can begin to redesign the schema. They state that "reverse engineering is all about creating [new] models from existing systems." (Sabanis and Darlison, 1992, p.124)

4.4 Data Modelling

4.4.1 Conceptual modelling.

The conceptual model is the most abstract type of data model describing a universe of discourse (Olle, 1993). The objective of a conceptual data model is to convey a significant amount of information in a readily understandable manner (Simsion, 1994). A conceptual model should not necessarily be immediately translatable into a DBMS schema, it should be totally independent of any Data Modelling paradigm (Olle, 1993).

The conceptual model then is a way of describing a universe of discourse that is meaningful to humans. It describes the structure of the specific universe of discourse[6]. The conceptual model in essence can be thought of as the general design plan for the universe of discourse (Nijssen and Halpin, 1989).

One type of modelling paradigm commonly used for conceptual modelling is the Entity-Relationship model. However it is unclear whether this model is used at the conceptual level or as a logical modelling tool (Atkins, 1995). Atkins goes further to say that the nature of the constructs used in the ER model do not lend themselves well to a conceptual modelling process.

The NIAM methodology is a tool used less commonly than the E-R method for conceptual modelling. NIAM's diagramming constructs allow the analyst to model a universe of discourse relatively unencumbered by requirements of particular database paradigms (Nijssen & Halpin, 1989).

Conceptual modelling can be done by anyone involved in a universe of discourse although it is likely that the novice designers are unaware of what they are doing. Also because their conceptual modelling is not formalised it is unlikely to be rigorous and thorough. So if a novice designer has built a database schema without a conceptual model how hard would it be to generate a model from the physical schema? If this can be done it can be put to a number of uses:

- confirmation that the actual design maps to the novice designer's view of the universe of discourse.
- a starting point for a database consultant to re-engineer an existing database if the original designer is no longer available.
- as a documentation tool.
- as the input to a CASE tool that will generate a new database schema.

---

[6]eg. what type of objects populate the universe of discourse, what roles these play, and what constraints are in effect

4.4.2 Logical modelling.

Logical modelling is the step between conceptual modelling and physical modelling. It is where the universe of discourse is described in the constructs required for a particular Data Model paradigm but irrespective of performance considerations. i.e. the logical model is constructed with the intention of implementation in a DBMS based on the relational paradigm.

Entity-Relationship modelling is perhaps the most common type of logical model used for RDBMS implementation. Others such as SSADM (Flynn & Fragoso-Diaz, 1993) and MERISE (Quang, 1986) are adapted from E-R modelling and developed as standards for use in particular environments.

4.4.3 Physical modelling.

This is a depiction of how the data involved in a universe of discourse will map onto a particular DBMS. It generally uses similar constructs to the logical model but includes arbitrary design decisions made usually in the interests of system efficiency. In relational systems this will include at least a description of the tables to be included, the attributes within each table, primary and foreign keys and attribute domains.

Physical models can be textual in nature as their purpose is primarily to provide a description or a map of what the database schema looks like. It is not necessary that the physical model convey semantic information about the universe of discourse, indeed because of the design decisions made for performance considerations the physical model may in fact prove confusing and misleading if used to elicit semantic information about the universe of discourse.

4.5 The NIAM Methodology

The NIAM methodology is described by its creators as "a fact oriented approach to conceptual schema and relational database design" (Nijssen & Halpin, 1989). The methodology arose from the creators' perception that database designers need more than a set of concepts and suitable modelling methods to create quality data models,

they require a systematic procedure. It is this rigorous development procedure that the creators' claim distinguishes the NIAM methodology from other conceptual modelling methodologies such as entity-relationship modelling and normalisation.

1. Transform familiar information examples into elementary facts, and apply quality checks.
2. Draw a first draft of the conceptual schema diagram, and apply a population check.
3. Eliminate surplus entity types and common roles, and check for derived fact types.
4. Add uniqueness constraints for each fact type.
5. Check arity and logical derivation of fact types.
6. Add object type, mandatory role, subtype and occurrence frequency constraints
7. Check that each entity can be identified.
8. Add equality, exclusion, subset and other constraints.
9. Check that the conceptual schema is consistent with the original examples, has no redundancy, and is complete.

Figure 8. NIAM Conceptual Schema Design Procedure (from Nijssen & Halpin, 1989).

This section gives an overview of the NIAM methodology by describing its conceptual schema design procedure (CSDP), the outline of which is shown in Figure 8. It emphasises those concepts most relevant to the present report. A complete and detailed description of the methodology's meta-concepts, graphical notation, design algorithms and heuristics is provided in Nijssen & Halpin (1989).

4.5.1 CSDP Step 1: From examples to elementary facts and quality checks.

Step one is the "foundation of NIAM's design procedure" (Halpin and Orlowska, 1992 p.97). The recommended procedure is to examine output reports and input forms to extract familiar examples and transform them into elementary facts depicting the universe of discourse. To exemplify how this might be done Nijssen & Halpin (1989) advocate using a telephone analogy, ie. being restricted to communicating structured information in a simple manner over a telephone line. To explain this Table 1 depicts a report showing the allocation of students to different tutorials and where and when those tutorials are held.

Tutorial	Time	Room	Student ID	Student Name
A	Mon 3pm	SST2.27	08456373	Jones FD
			92345678	Smith KJ
			91234567	Olds BG
			90876543	Harris GH
B	Tue 2pm	SST2.22	90258456	Goodwin SA
			91245689	Smith CD
			92458741	George CA

Table 1. Tutorial Output Report.

The idea expressed with this example is that when information is verbalised people can express it concisely by stating a series of simple facts as shown in the following list:

- Tutorial with group name A meets at Time Mon 3pm
- Tutorial with group name A is held in Room with room number SST2.27
- Student with student # 08456373 belongs to Tutorial with group name A
- Student with student # 08465373 has Name 'Jones FD'

Verbalisation of this type requires some interpretation of the data therefore it is important that the verbaliser be familiar with the universe of discourse. The above list is composed of what are termed elementary facts, ie. they are sentences that cannot be reduced into smaller sentences which when combined provide the same information as the original. Defined more formally *'an elementary fact asserts that a particular object has a property, or that one or more particular objects participate together in a relationship'* (Nijssen & Halpin, 1989 p.36). Facts exhibiting the same structure but containing different values are grouped into aggregations called fact types

Elementary facts consist of up to four constructs; objects, predicates, reference modes and labels. Each of these constructs is identified here by restating the tutorial example above: *"**Tutorial*** (object) with ***group name*** (reference mode) ***A*** (label) is ***held in*** (predicate) ***Room*** (object) with ***room number*** (reference mode) ***SST2.27*** (object)".

Once a set of example sentences has been translated into elementary facts two quality checks are performed before moving to CSDP step 2. The first check requires that the objects in the elementary facts are well defined. The second, that the elementary facts are unable to be split into simpler facts without loss of information.

The first step in the CSDP, has received criticism (Darke & Shanks, 1995) on the grounds that as a requirements definition process it does not adequately address knowledge elicitation that arises from social interaction. By suggesting that example sentences are composed by the expert in the universe of discourse, and that these sentences form the basis of the system's requirements, the CSDP process requires that all the necessary information is predefined (ie. found in forms and reports). Gougen (1992) suggests that there are also emergent requirements that only come into existence as a result of the combination of the knowledge about the universe of discourse, held by the users, and knowledge about the technical possibilities, held by the analyst. Darke & Shanks(1995) state that the process used in the CSDP offers no latitude for the inclusion of these emergent requirements because its formalism precludes interaction between the analyst and the user. At this time the author has seen no direct refutation of these criticisms

4.5.2 CSDP Step 2: First draft of conceptual schema diagram and population check.

The second step in the CSDP takes the completed set of elementary facts from step one and renders them as a conceptual schema diagram. The diagram consists of conventions describing each construct of the elementary fact types identified. An object type (entity type) is shown as a named ellipse. A label type is shown as a named broken ellipse. Predicates (roles) are shown as boxes placed between two ellipses. The predicate text is placed either inside or adjacent to the relevant role box. Reference modes are usually placed textually in parentheses in the relevant ellipse.

Figure 9 shows a conceptual schema diagram for the elementary facts expressed below.

- The person with name 'Ryder M' drives the car with reg # 'KL700'.
- The person with name 'Ryder M' drives the car with reg # 'MQ3601'.
- The person with name 'Atkins S' drives the car with reg # 'KL700'.

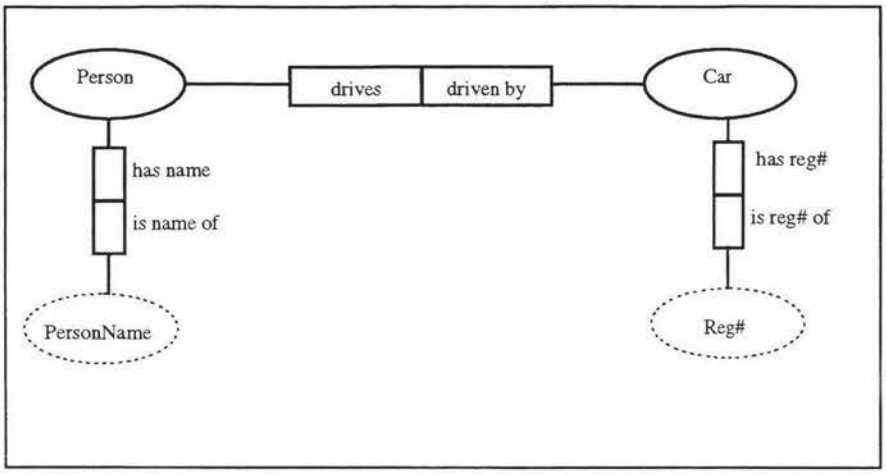


Figure 9. A Conceptual Schema Diagram (Nijssen & Halpin, 1989, fig 3.5).

Sometimes the description of a label and a reference mode will be the same. In this case the label will not be shown explicitly on the diagram but the reference mode will be included enclosed in parentheses with the entity type as shown in Figure 10.

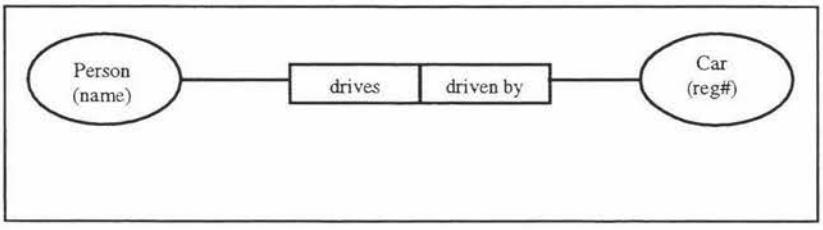


Figure 10. Schema Diagram with Reference Modes in Parentheses (Nijssen & Halpin, 1989, fig 3.6).

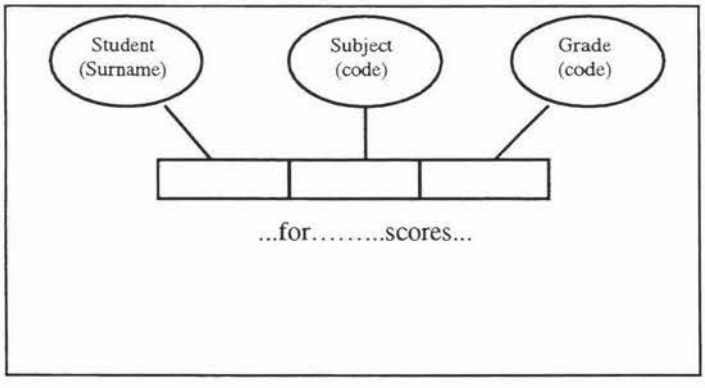


Figure 11. A Ternary Fact Type (Nijssen & Halpin, 1989, fig 3.11).

The fact type shown in Figure 10 is a binary fact type. Many data relationships however are too complex to be depicted using binary fact types. Conceptual schema diagrams provide capabilities for depicting these relationships by using an extension of the predicate box to extend the arity of the fact type (Figure 11) or objectification of a predicate to model nested relationships (Figure 12).

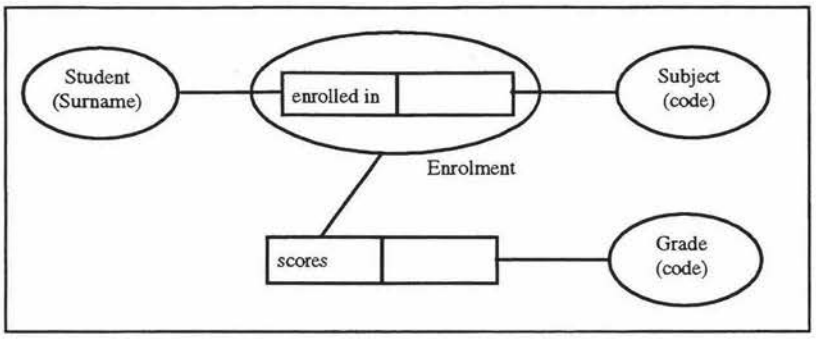


Figure 12. A Nested Fact Type Using Objectification (Nijssen & Halpin, 1989, fig 3.12).

4.5.3 CSDP Step 3: Eliminate unnecessary schema and find derived fact types.

Having produced a conceptual schema diagram representing the universe of discourse, step three requires a critical examination of the diagram. The objective of this examination is to identify and eliminate common entity types and derived fact types. The presence of common entity types is undesirable because they represent a degree of ambiguity in the definition of entities and they unnecessarily clutter the diagram. A clue to the presence of common entity types is that they often have the same unit-based reference mode. Figure 13 and Figure 14 show an example of common entities being reduced to a single occurrence.

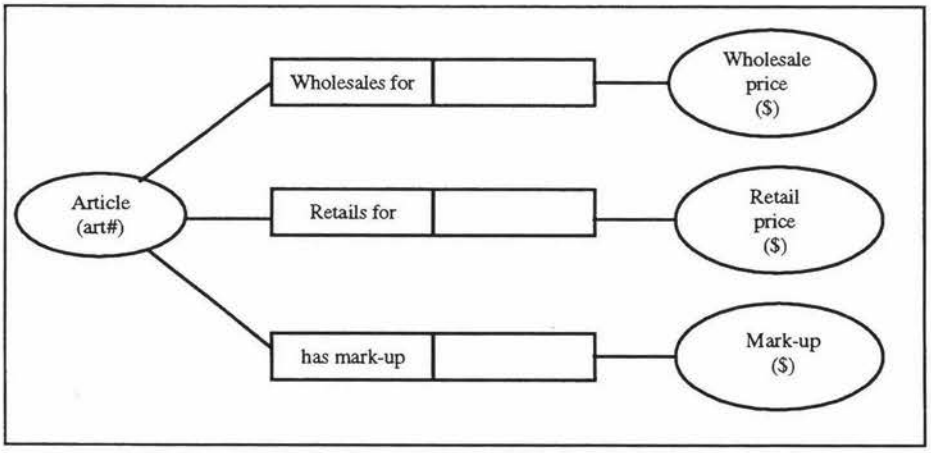


Figure 13. A Faulty Conceptual Schema (Nijssen & Halpin, 1989, fig 3.13).

Derived data is something found in many systems (Ricardo, 1990) and the choice of strategy to use concerning its management is often left until a later stage in the development of the system. It is however desirable to be able to identify where derived data exists on a diagram. In the NIAM conceptual schema diagram this is done with an asterisk as shown in Figure 14 where the mark-up can be derived by subtracting the wholesale price from the retail price.

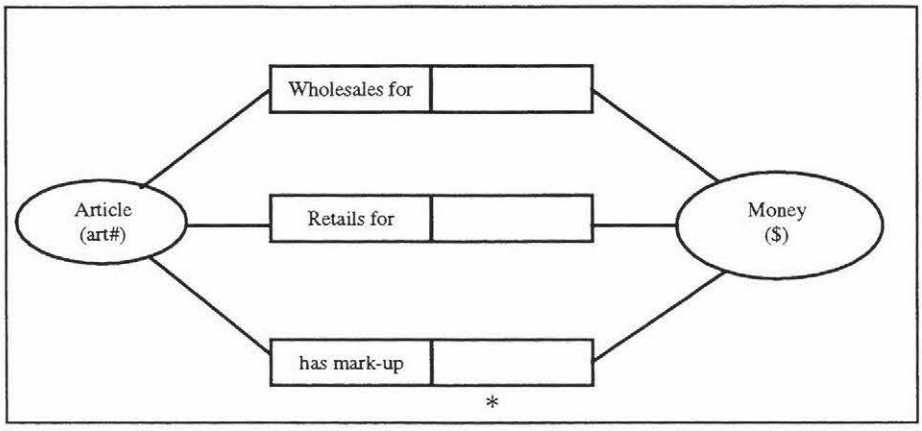


Figure 14. Step Three Applied to Figure 13 Including Derived Data (Nijssen & Halpin, 1989, fig 3.14).

4.5.4 CSDP Step 4: Add uniqueness constraints.

This step focuses on the addition of static constraints to the roles identified in the conceptual schema diagram. The step explicitly dwells on uniqueness constraints because it is these that play a pivotal role when the conceptual schema is later mapped onto a relational database (Nijssen & Halpin, 1989).

Uniqueness constraints are specified on a diagram with a double ended arrow spanning the role or roles that must be unique in that fact type. Every fact type must have at least one uniqueness constraint and may have more than one. Derived fact types strictly speaking do not require the notation of a uniqueness constraint as this can also be derived from the uniqueness constraints associated with the fact types from which it is derived. However for exposition purposes it is desirable to include these on a diagram.

Figure 15 shows uniqueness constraints applied to unary and binary fact types in one diagram. A unary fact type is a fact type that specifies only one property of an object (Nijssen & Halpin, 1989). This diagram conveys the message that according to the binary fact type each combination of person and subject must be unique. This is one of four variations of uniqueness constraint for binary fact types as shown in Figure 16. The uniqueness constraint on the unary fact type 'Jogs' is the only possible variation of constraint and means that each instance of person must be unique.

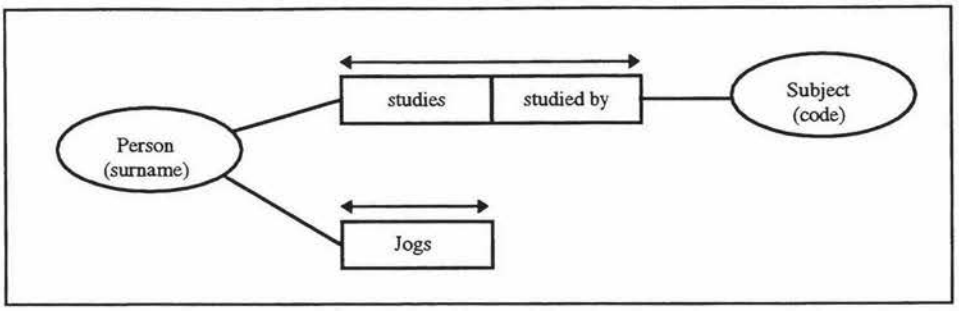


Figure 15. Unary and Binary Uniqueness Constraints (from Nijssen and Halpin, 1989).

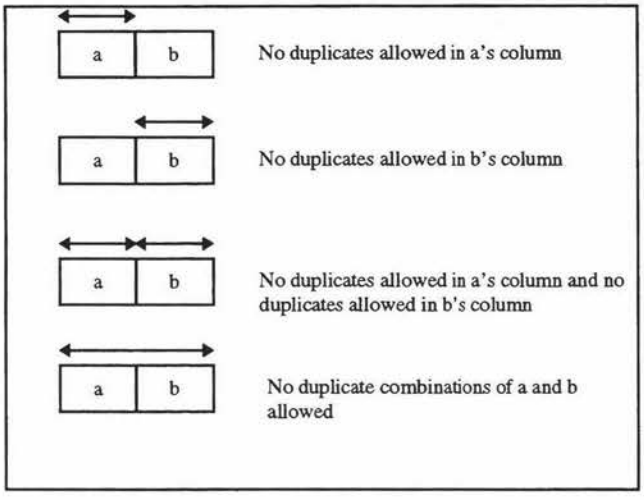


Figure 16. Binary Uniqueness Constraint Patterns (from Nijssen and Halpin, 1989).

Fact types of longer than binary length have a larger set of possible uniqueness constraints and may simultaneously have more than one uniqueness constraint. However the most common fact types are binary in nature and each higher arity becomes less common (Nijssen & Halpin, 1989). Figure 17 shows the set of legal constraint combinations for ternary fact types.

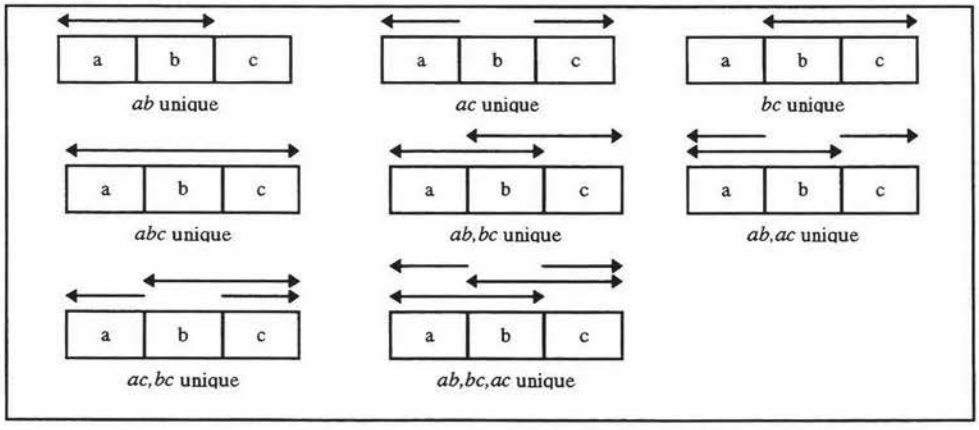


Figure 17. Ternary Fact Type Constraint Combinations (from Nijssen and Halpin, 1989).

For a complete discussion of objectified fact type constraint variations and higher arity fact type constraint variations see chapter four in Nijssen & Halpin (1989).

4.5.5 CSDP Step 5: Arity and logical derivation checks.

In the fifth step the analyst is required to check the results from the previous steps to ensure that there are no fact types that are either too long or too short. When fact types are too long it means that they can be reduced into more elemental fact types without losing any information. Fact types that are considered too short indicate that reduction to the current arity has resulted in loss of information. If the preceding steps of the CSDP have been completed carefully it may be that all fact types are the appropriate arity and therefore this step will contribute nothing to the design, but it will instil confidence in its quality. The following three methods (Nijssen & Halpin, 1989) are suggested for checking the appropriate arity of fact types

1. Use your common sense or background knowledge of the universe of discourse to decide if information is lost by splitting a fact type (can the fact type be expressed as a conjunction of smaller ones).
2. Use formal splittability rules about the shortest key[7] .
3. Provide a significant fact table for the fact type, split this by projection then recombine by natural join: if new instances appear then the fact type is unsplittable in this way. Test until a split is found or all ways are exhausted.

(Nijssen & Halpin, 1989)

Rigorous application of these checks will produce a conceptual schema diagram sufficiently refined and of high enough quality to move to step six.

4.5.6 CSDP Step 6: Add additional constraints.

This step is concerned with adding entity type, mandatory role, subtype and occurrence frequency constraints to the conceptual schema diagram. Figure 18 shows how an entity type constraint is depicted on a diagram. Here the set of possible types of 'Medal' is G, S and B and the minimum and maximum number of medals winnable

[7] For a complete description of splittability rules see chapter five of Nilssen & Halpin (1989).

ranges from 1 to 200. These constraints are portrayed on the conceptual schema diagram by placing the range of values within sets of braces.

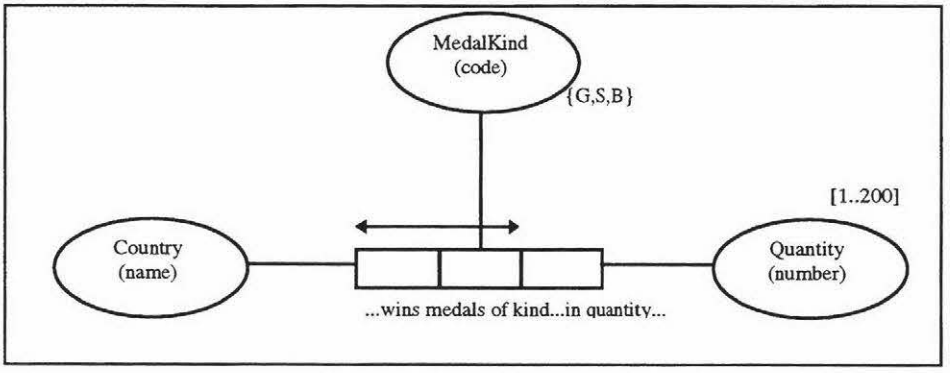


Figure 18. Conceptual Schema Diagram with Entity Type Constraints (Nijssen & Halpin, 1989, fig 6.4).

Mandatory and optional role constraints are depicted by placing a mandatory role dot on the entity type attached to the role in question. Mandatory roles can be shown explicitly for all fact types as shown in Figure 19. Alternatively the implicit mandatory roles can be left un-dotted producing a cleaner diagram and attracting attention to those mandatory roles that are most important as shown in Figure 20.

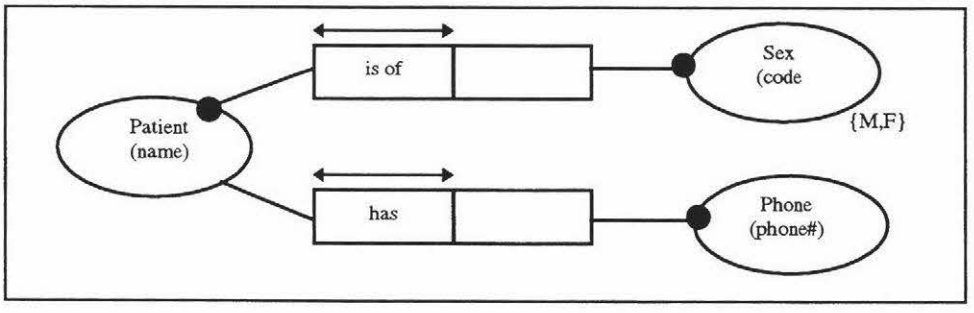


Figure 19. Explicit Mandatory Role Constraints (Nijssen & Halpin, 1989, fig 6.6).

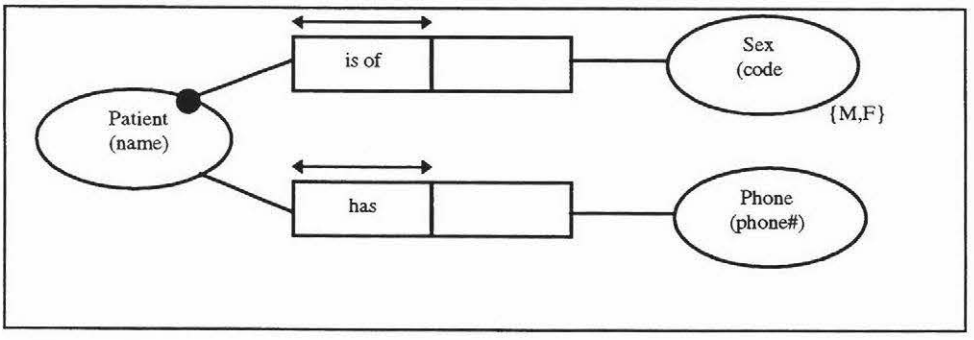


Figure 20. Implicit Mandatory Role Constraints (Nijssen & Halpin, 1989, fig 6.7).

The next constraint type to identify and include in the conceptual schema diagram is the subtype and super-type constraint. This constraint is depicted by an arrowed line

as shown in Figure 21 connecting the two entity type depictions. In this example the entity types Man and Woman are subtypes of Person.

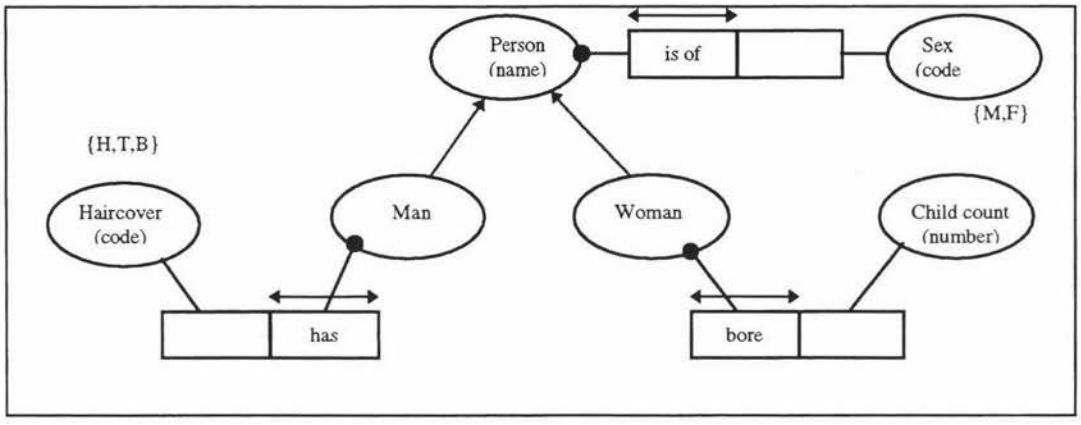


Figure 21. Conceptual Schema Diagram with Subtypes (Nijssen & Halpin, 1989, fig 6.23).

The last constraint type addressed in step six is occurrence frequency constraints. This refers to the number of times a given entity type may be recorded playing a given role.

A uniqueness constraint is a special variation of a frequency constraint and is denoted on a schema diagram with a double headed arrow rather than the inclusion of a number above or below the relative role. Figure 22 shows a schema diagram with a frequency occurrence constraint of three attached to the year role. This means that one row is required containing the quantity sold and the year sold for each type of drink. An example table is included to further clarify the data that might be stored complying with this constraint.

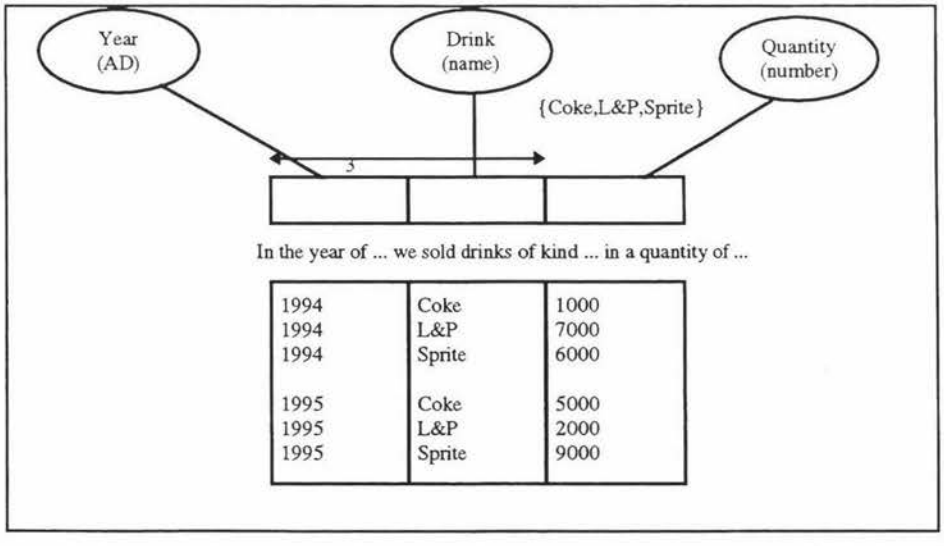


Figure 22. Schema Diagram with Frequency Constraints(Nijssen & Halpin, 1989, fig 7.10).

Frequency constraints can also be placed across multiple roles and include ranges of frequencies[8] relating to a role. For more detailed explanations of these constraints see chapter six of Nijssen & Halpin (1989).

4.5.7 CSDP Step 7: Entity identification.

This step requires that each entity be examined to detect which fact types identify the entity. Each fact type that contributes to the identification of the entity must have a mandatory constraint placed on it. If this involves more than one fact type an inter-reference type uniqueness constraint depicted by a 'u' within a circle connected to respective roles by a dotted line is added. This is shown on the diagram in Figure 23.

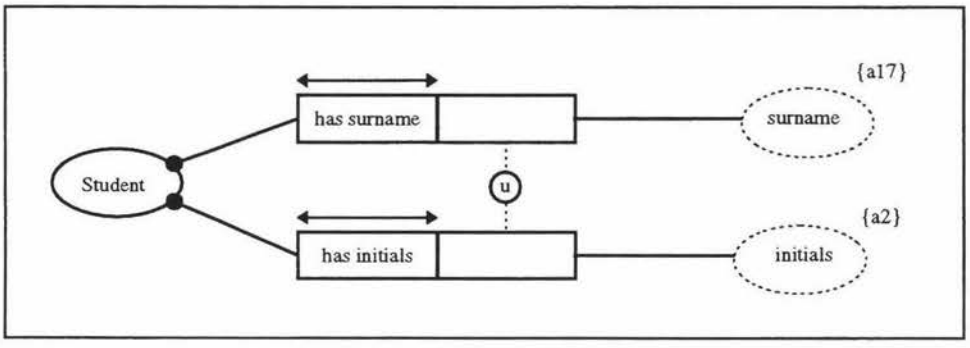


Figure 23. Entity Identification Requiring Two Labels (Nijssen & Halpin, 1989, fig 7.11).

This identification is often done in earlier steps, however completion of this step gives added confidence of the quality of the conceptual schema design and clarifies understanding of the universe of discourse. Note that this step applies to the relationships between entities and their labels. Once the entity identification has been determined it is usual to contract Figure 23 to its equivalent shorthand version as shown in Figure 24.

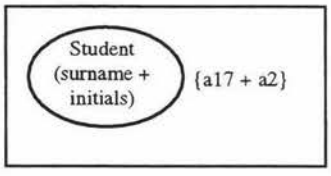


Figure 24. Shorthand Notation for Figure 23.

[8]These are stipulated with minimum and maximum numbers above or below the role.

Label descriptions are also included in Figure 23 where for surname "{a17}" means an alpha-numeric string of up to 17 characters. This label description is concatenated and included in Figure 24 as {a17 + a2}.

4.5.8 CSDP Step 8: Add additional constraints.

Step eight extends the range of possible constraints that can be applied to the conceptual schema diagram. The constraints described in earlier steps cover the vast majority of constraints that are of practical significance, however there are times when other constraint types need to be applied. The types of constraints introduced in this step are constraints applied to homogeneous binary fact types and equality, exclusion, and subset constraints. A homogeneous binary fact type is one where both roles are played by entities of the same type as shown in Figure 25.

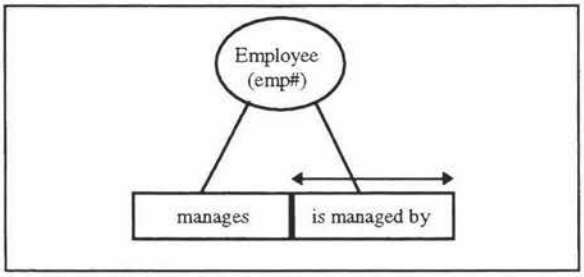


Figure 25. Homogeneous Binary Fact Type (from Nijssen & Halpin, 1989).

Constraints covered in this step are not described any further in the present report as their high level of complexity and relative obscurity would require a significant amount of time and space to cover methodically. An explanation of these can be found in chapter eight in Nijssen & Halpin (1989).

4.5.9 CSDP Step 9: Completeness and succinctness checks.

During the final step of the CSDP no additional constructs are added to the diagram produced at the end of step eight. This step is a checking step. The analyst is advised to go back to the beginning and check that the current diagram is consistent with the data and information about the universe of discourse. Specifically there are three checks to be done; a population check, a redundancy check and a completeness check.

The population check involves building a table of representative data for each fact type as shown in Figure 22 and checking that the sample population agrees with the constraints imposed over the preceding eight steps. The redundancy check requires examination of the conceptual schema diagram for two types of redundancy; stored redundancy and derived redundancy. Stored redundancy is the simplest to identify and occurs when either a fact type is stored more than once or when a fact type of higher arity than three can be decomposed without loss of information. Finally the completeness check requires that semantic information from the universe of discourse be recorded and then a check be made against the diagram to ensure that each piece of semantic information is represented in the diagram. A common code is annotated on each list item and in the appropriate place on the diagram to ensure that the check is complete.

Provided the preceding nine steps are conscientiously followed Nijssen & Halpin (1989) claim that the methodology will produce a high quality conceptual schema giving a thorough and accurate representation of the universe of discourse. This conceptual schema can then be transformed into a Data Model dependent schema and implemented on a DBMS.

4.6 Mapping from conceptual to implementation models.

One of the primary objectives for creating conceptual models of a universe of discourse is to use them to produce implementation database schemata. Although strictly speaking the implementation schema is produced by imposing physical considerations on a logical model, which is derived from the conceptual model, in this section of the report we will treat the implementation and logical models as though they were the same. A conceptual model can be mapped to an implementation schema for any database paradigm considered suitable for the universe of discourse. While recognising the presence of other database paradigms this report directs its attention to the production of implementation schemata for the relational paradigm.

An implementation schema should efficiently model the subset of the real word that it describes (Nijssen and Halpin, 1989). A model by definition simplifies the real word by concentrating on the important and relevant points and ignoring those aspects that are

irrelevant. In database terms the data that is stored in the implementation schema should consist only of data that is necessary to enable a system to describe, record or control aspects of the real world relevant to that system.

Human understanding of the real word is also dependent on modelling, and as with modelling that understanding too is a simplification of the real thing (Johnson-Laird, 1983). A novice designer is unlikely to use any formal data centred conceptual modelling method but nevertheless will generally have a mental model of the universe of discourse. Mapping this mental model to an implementation model will prove to be a difficult task in any but the simplest of situations. The novice designers problems are compounded by a minimal appreciation of the theory of the paradigm into which they are mapping their model. As there are a large number of ways even a modest database schema can be designed it is difficult for the novice or experienced designers to objectively judge the quality of their designs.

Using Entity-Relationship modelling as a conceptual design tool produces a more formal data model that can then be mapped to an implementation schema using established guidelines. Unfortunately the guidelines provided for mapping an E-R model to a relational schema also requires an amount of interpretation by the designer (Ricardo, 1990). Normalisation is often used as a relational design tool to improve the quality of the schema resulting from mapping an E-R model to a relational paradigm. Depending on the ability of the designer the resultant implementation schema is likely to be of adequate quality, but as Atkins (1995) notes, due to the nature of the process the resultant schemata is not reproducible[9] and there is still no way for the designer to objectively assess the quality of the implementation schema as it relates to the universe of discourse.

The NIAM methodology claims to deliver a more reproducible implementation schema from its mapping process (Nijssen & Halpin, 1989). Because the conceptual schema typically contains more semantic content than an E-R model there is less interpretation

[9]Any two implementation schemata mapped from the same conceptual model are unlikely to be exactly the same.

required by the designer when mapping to an implementation schema. Nijssen & Halpin (1989) claim that the logical relational model produced from a complete conceptual schema using the NIAM methodology is completely reproducible. This claim of reproducibility is supported by the publication of the optimal normal form algorithm (ONF) described in the following section.

4.7 Optimal normal form algorithm

A detailed description of the optimal normal form (ONF) algorithm is given in Nijssen & Halpin (1989, pp 254-279). The following narrative draws almost exclusively from that work and summarises it within the context of this report.

The ONF algorithm provides a systematic procedure for grouping the fact types of a conceptual schema into a set of tables that when combined make up a relational schema. This is the first of two stages required to map a NIAM conceptual schema into a relational DBMS application. The second stage involves writing procedures to implement additional constraints and derivation rules that reside outside the RDBMS. The second stage is concerned mostly with application design rather than database design and is consequently outside the scope of the present report.

The ONF algorithm uses the relational model as its target data model because it offers simple data structures and simple operations. The aims of the ONF algorithm are to produce a redundancy free design using a minimum number of tables. The ONF algorithm typically produces tables in fifth normal form with a minimal number of tables in the overall schema. This is why the algorithm is described using the term "optimal".

Nijssen & Halpin (1989) claim the algorithm is not complex. It has been used commercially to produce relational schemata in the NIAM based CASE tool Infomodeler[10]. The algorithm requires that each fact type map into only one table, with roles mapping into columns. If a fact type maps onto a single table (each row

[10] InfoModeller is a registered trademark of Asymetrix.

contains only one fact) then all its keys (uniqueness constraints) map onto keys of that table. The first step to creating a relational design is to group the fact types in the NIAM type diagram into the following three groups and map them accordingly:

1. For each fact type without a simple key, create a separate table. Select a shortest key of the fact type as the primary key of the table.
2. Group fact types with simple keys attached to a common object type into the same table with the primary key based on this object type.
3. For each remaining fact type, create a separate table. Select a key of the fact type as the primary key of the table.

Choosing table and attribute names should be done carefully striving to use meaningful names where ever possible. Often fact type names can be used as table names and role names used for attributes. Other times it may be necessary to use other names.

Having grouped the fact types and translated them into tables and attributes, the next step is to distinguish the primary key, candidate keys and foreign keys. The following examples, adapted from Nijssen & Halpin (1989), illustrate and clarify this process.

According to ONF step one, all fact types without a simple key are mapped to a separate table as shown in Figure 26.

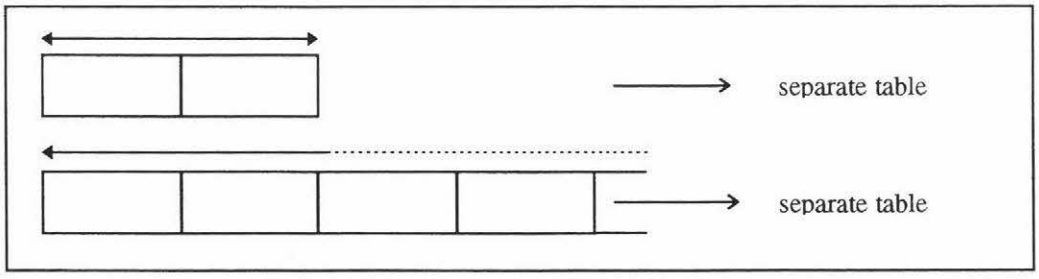


Figure 26. Composite key mappings. (from Nijssen & Halpin, 1989)

Step two of the ONF algorithm states that where an object type has more than one fact type attached all of these fact types should be included in a single table with the common object type acting as the primary key for that table. Figure 27 shows object type A involved in two fact types. This is mapped onto a ternary table with A used as the key value.

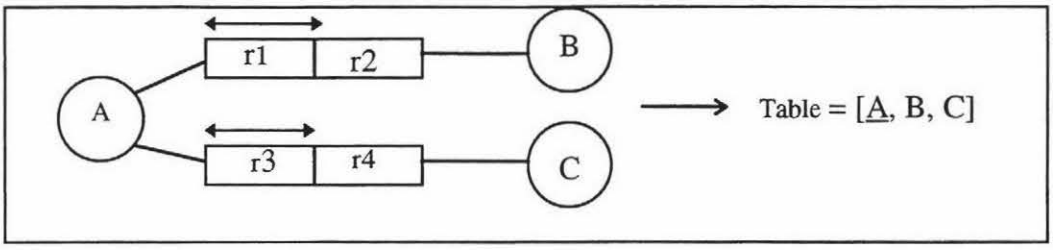


Figure 27. Grouping fact types to one table (from Nijssen & Halpin, 1989).

The third step in the ONF algorithm is translate the remaining fact types into separate tables. Because these fact types have not been dealt with in ONF step one or two they must be either binary facts with a simple uniqueness constraint or unary fact types. Figure 28 shows an example where the right hand fact type (awarded) would be translated using step one and the remaining fact type (works for) using step three. Note that there is an equality constraint[11], shown as a dotted line, between the Lecturer fields of both tables. This indicates that both roles played by Lecturer are mandatory.

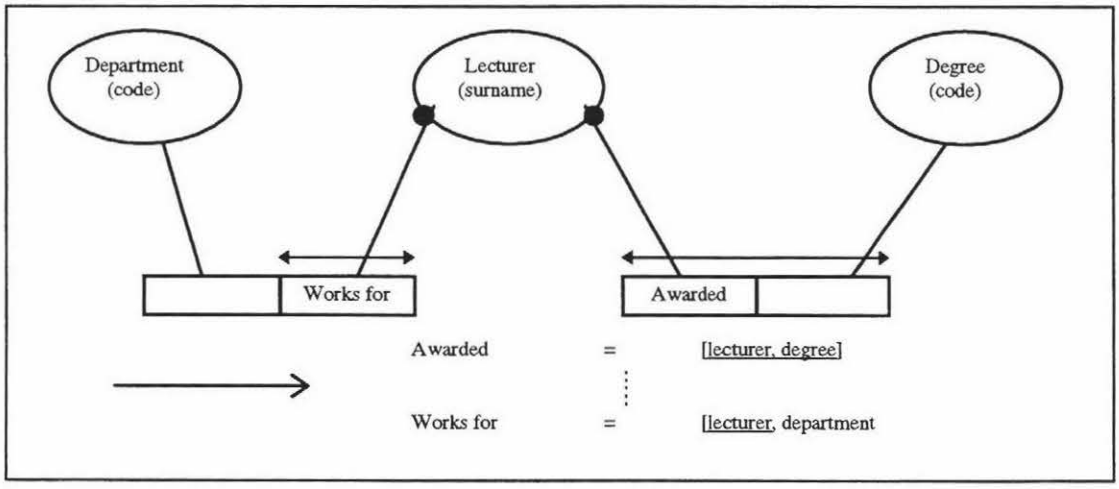


Figure 28. Mapping Using ONF Step Three (Nijssen & Halpin, 1989, fig 11.7).

These three steps provide the nexus for transforming a NIAM conceptual schema into a relational schema in optimal normal form. There are many complexities that can be included in a conceptual schema that require special considerations for translation

[11]Equality constraints are covered in step eight of the CSDP. See Nijssen & Halpin (1989) pp 173 and 253 for a more detailed description of this constraint type.

purposes. These considerations are discussed in detail in chapter 11 of Nijssen & Halpin (1989).

The potential explored in the present report concerns a reverse engineering version of these three steps into a reverse data modelling procedure to produce a NIAM conceptual diagram from an existing implementation schema. Because the process will produce a predictable conceptual model it is then possible to refine this model against the universe of discourse and re-process it through the ONF algorithm to produce an optimally normalised implementation schema. The complete reverse data modelling procedure is covered with examples in chapter seven of the present report.

SECTION THREE

SURVEY

5 Σ Designer survey

This chapter describes the method, results and interpretations pertaining to a survey of a sample of novice designers undertaken for this thesis. The purpose of the survey is to support assertions made in the body of this thesis and investigate the level of commonality among the respondents' experiences. The first assertion this thesis makes is that the class of Σ database designer as described in the report does have a presence in the community. This area is covered by questions at the start of the questionnaire. The second assertion is that there is little help offered by product documentation and DBMS vendors in the area of designing a database structure. Respondents' experiences of this are covered in later parts of the questionnaire. The third assertion concerns the extent to which the stability, extensibility and evolution of database structures designed by novice designers meet the system's on-going functional requirements. While the questionnaire provides information about the stability, extensibility and evolution, judgement of the database structure's ability to meet on-going requirements is deferred to case studies arising from the survey.

5.1 Sample selection

Identification of an available sample population of novice designers presents a challenge when researching the activities of these people. By definition novice designers are not likely to be grouped by any common socio-demographic factors such as involvement in a particular industry sector or membership of a society, however a possible exception might arise from their interaction with software vendors as all will have, or have had, access to RDBMS software which must have been acquired at some time. Therefore a sample from software vendors' customer lists could provide a useful survey population. However software vendors approached were unwilling to cooperate with the author to the extent of permitting their customer data to be used as a survey population. They did instead agree to allow the author to use some space in an issue of their monthly customer newsletter to elicit responses from novice designers.

Whilst eliciting responses using this method is likely to identify a number of novice designers it also places restrictions on the extent and validity of statistical sampling performed on the survey's results. Being aware of this statistical caveat and yet wishing to spread as wide a net as practical to contact novice designers, five additional organisations producing similar publications were approached of which three agreed to provide sufficient space. Articles consisting of, or similar to, the text in Figure 29 appeared in the following publications: a monthly newsletter of a computer products retailer reaching approximately 800 customers; a monthly newsletter published by a city's Commerce Centre circulating to approximately 1,500 people; a weekly university staff newsletter circulated to the university staff of about 1,000; a business supplement in a local newspaper with a circulation estimated around 20,000.

Research help wanted.

Database packages for PCs such as Paradox, Access, DBase and others have become readily available for business and individual use in recent times. However there is no commonly held metaphor to assist people in grasping the concepts required to implement quality database designs in their business area.

The identification of strategies to assist the typical business person to use these products effectively is the issue addressed by research being undertaken by Mike Ryder for the Information Systems Department at Massey University.

If you've had an encounter with a database system of this type and are willing to participate in a survey, please phone 350-xxxx(wk) or 357-xxxx(hm) or drop a line to Mike Ryder C/- Information Systems Department, Massey University.

Figure 29. Request for respondents

In addition to the sample identification strategy mentioned previously, questionnaires were sent to a mailing list of large organisations. These organisations are a subset of organisations contained in a mailing list used in previous, unrelated research. The criterion used to select which organisations to mail to, is that an organisation be included if the nature of that organisation is likely to contain novice designers. Because little is known about many of the organisations, selection required taking educated guesses at which organisations were likely to contain autonomous business units or foster the type of organisational culture that will provide novice designers the

conditions their activities require. Questionnaires were sent to a final mailing list of 97 addresses selected from 177 potential addresses contained in the larger list.

5.2 Survey response rate

The advertising strategy elicited nine responses of which seven fitted the definition of novice designer as described in the present report. Of these seven, three are documented as case studies in this thesis.

A total of 104 questionnaires were sent, from which 40 replies were received giving a total response rate of 38%. Questionnaires sent unsolicited to the companies on the mailing list totalled 97, from which 33 responses were received. From these 33 the results of 22 are included in this report. Those responses that are not included are omitted because they are either significantly incomplete or the respondent's answers strongly suggest they are not a novice designer. The seven questionnaires sent to the self-selected survey population received a 100% response rate, however one is omitted due to significant incompletion consequently only the remaining six are included in the results. This gives 28 valid responses, five non-novice designer responses, seven invalid responses and 64 non-responses from the total of 104 questionnaires as shown in proportional terms in Figure 30.

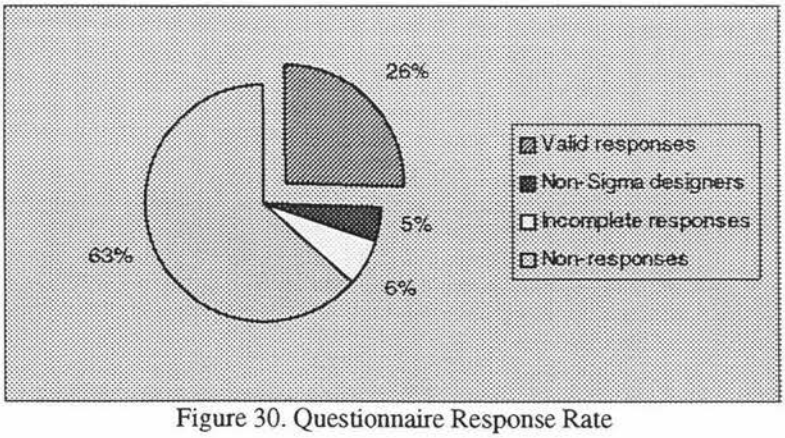


Figure 30. Questionnaire Response Rate

The strategy of eliciting responses from people who consider themselves to be novice designers through advertising has produced more expansive and superior quality responses than information obtained from novice designers identified using the mailing list. The elicitation strategy produced only 15% of the responses and proved more

time consuming and complex than the mailing list which produced the remaining 85 % of responses.

5.3 Questionnaire structure

The questionnaire comprises three sections and a cover page (Appendix 2). The cover page asks recipients to participate only if they have designed one or more databases and consider themselves not to be an information systems professional. The first section, titled database involvement, questions the extent of the respondent's exposure to DBMS and other computer products primarily in the course of their work. Questions in this section look first at the respondent's work place, the proportion of their time spent using computer products, their position within the organisation and establishing the organisation's size. After establishing their computing proficiency respondents are asked what generic applications they use regularly and in particular what database packages they have used and the extent of their experience in designing database schemata.

The reason for selecting questions in these areas is to establish that the respondent fits loosely into the definition of a novice designer thereby supporting the existence of such a class of people as asserted in this report. It also provides a sample of vocations where database technology can be, and is being used.

Section two looks at which database products are used by respondents, how these products are used and to which business functions they are applied. It investigates the scope of each respondent's most recent database by asking for the approximate number of tables in the database and the number of rows in the largest tables. This gives an indication of the database's complexity. Questions in this section also deal with the extent to which the database application is used within the respondent's organisation by asking who uses it and how often it is used.

Section three investigates issues raised during the process of database design and some implications for the evolution of that design to accommodate changing functional and data requirements. These questions look at the extent of each respondent's knowledge of database issues and from where that knowledge was acquired. Several database

design strategies are proposed and the respondent is asked which best suits the process followed during the design of their database. The respondent is then asked how much the database schema has changed since its original design and asked to assess issues concerning the future of their database and strategies they might employ to manage its evolution.

5.4 Discussion of survey results

The results of this survey can be used to postulate hypotheses regarding several aspects of database usage by novice designers. This section of the report deliberately accentuates those results that directly relate to the assertions stated at the beginning of this chapter. Other possible deductions emanating from the results, are included as issues of secondary interest.

The three questions included on the cover page of each questionnaire are designed to minimise the number of responses from Ω and Φ database designers. These questions identify those candidates that are suitable respondents by asking whether the intending respondent:

1. uses a relational database management system,
2. has built tables or an application with a relational database management system,
3. considers themselves to be other than an Information Systems professional.

Questionnaire recipients' were asked to pass the questionnaire to someone else if they could not answer yes to each of these questions. Despite this instruction a number of respondents provided answers that indicated they were likely not to be novice designers. Six respondents gave an answer in excess of ten databases to question eight in section one that asks how many databases the respondent has designed and built. For most novice designers the number of databases designed is likely to be less than ten. Removing five of these respondents from the statistics decreased the average number of databases built from 13.25 down to four. In addition to question eight in section one, questions one to four in section three also provide indications of how well the respondent's experience fits with the definition of a novice designer. Five of the respondents' answers strongly suggest that they are not novice designers and answers

from three others indicate that they are borderline cases. Consequently based on a combination of responses to these questions the author has excluded five questionnaire responses from the results used in this report and found in appendix one.

It is assumed from the combination of answers given that the remaining 28 responses are from genuine novice designers. Provided this assumption is valid the survey provides evidence that novice designers do exist in the community. This validates the report's first assertion. However, due to the nature of sample selection used, the survey does not provide a useful indication of the prevalence of novice designers in the community.

A list of reported employment positions is recorded in Table 2 showing a range of job titles of those using DBMSs at the novice designers' level.

Accountant	Environmental Officer	Museum Director
Admin Superintendent	Forest Management	Personal Assistant
Analyst	Health & Safety	Parks Asset Management
Assistant Treasurer	Help Desk	Programmer
Corporate Accountant	Intelligence Analyst	Reference Librarian
Development Executive	Information Dev Officer	Systems Administrator
Director	Library Manager	Systems Manager
Electronics Technician	Market Analyst	Team Leader

Table 2. Reported Job Titles.

Assessment of the level of database design assistance offered by DBMS vendors to novice designers is rated from answers provided to questions two to six in section three. Analysis of the responses to questions two and three in section three shows that although 75% of respondents report having attended some type of database course only 40% of these found the course useful when designing the structure of their database. This suggests that some courses do not cover database design skills in an effective manner.

Respondents reported using a mixture of techniques for selecting the tables to include in their database. Notably no-one responded in the affirmative to the on-line help option, however many respondents grouped either formal process and expert advice, or intuition and trial & error together as their design process. The results indicate that

there is no preferred table identification method with no option being selected by more than 30% of the 28 respondents.

In addition to the findings from question four almost 60% of responses to question five indicate that the respondents found the DBMS's manuals to be little or no help with database design issues. While it is recognised that there might be variations in quality between different vendor's manuals the results from these two questions suggest that there are limited quality resources readily available to novice designers that can assist them with database design issues.

The remainder of the questions in section three are included to investigate the validity of the third assertion. Analysis of the results provided to these questions seems to suggest that respondents' database schemata have had little trouble supporting changing functional requirements. The majority of respondents' reported minimal structural changes and only minor on-going effects. If these responses are accepted at face value they contradict the assertion that novice designers' database designs are largely inflexible and unable to readily adapt to meet changing functional requirements. However the author contends that this issue is too complex and contains too many variables to be accurately judged using a written survey to survey a sample population. Factors such as the ability of novice designers' to assess these areas, and how respondents' have reported on schemata that have not undergone significant change may have resulted in responses giving false impressions. Instead this questionnaire was used as a means of identifying four novice designers willing to participate in a more thorough interview addressing this issue. Details and results of these interviews can be found in chapter 6 and appendix three of this thesis.

Despite deferring a conclusion regarding the third assertion, the results from questions six to nine in section three are revealing in terms of how respondents view the stability of their database designs. Slightly in excess of 70% of respondents report that the current database design is similar to the one with which they started. It may be argued that change in a database's design occurs slowly and usually only in response to functional changes. As 65% of the respondents have been using a DBMS for 3 years or less it is possible that many of these databases have not been in use for long enough to require non-trivial changes. Taken with the results from question eight where 90%

of those responding indicate that structural changes they have made have caused relatively few additional problems it seems more plausible that many of the respondents have not yet had occasion to attempt a non-trivial modification to their database schema.

The questionnaire also includes a number of questions not specifically referred to so far in this chapter that either reinforce the questions targeted toward verification of the assertions or provide a view of the environment within which novice designers work. These are discussed here as an adjunct to the main emphasis of the report to highlight any similarities between novice designers' responses.

- It would seem reasonable to expect novice designers to spend a large proportion of their time using computers. This was supported with 65% of respondents reporting that they spent between 61% and 100% of their time interacting with computers.

- If we accept that database concepts are not as widely understood as some other computer related concepts it would be likely that novice designers should possess good computing skills. This is upheld in the results with 86% of respondents rating their computer skills as good or very good.

- Considering that most novice designers are likely to have strong computing skills it is also likely that they will have had experience with other types of software packages. The survey supports this with 100% of the respondents regularly using three or more different types of software package, 60% using four or more, 37% using five or more and 7% using six or more.

- As novice designers can be expected to have designed and built a minimal number of databases it is likely that they will have had exposure to few DBMSs. This is born out by the survey results with 63% only having used one DBMS, 89% having used no more than two and no-one reporting to have used more than three.

- Another finding in the survey results is the high popularity of Windows based DBMSs especially Microsoft Access. 64% of respondents reported using a Windows based DBMS and of these, 89% were using Microsoft Access.

- There is an even distribution of responses between the options of constantly, hourly, daily, weekly and monthly regarding the regularity with which the respondent's database is currently used. This indicates that novice designers apply database solutions to situations of varying immediacy.

- 75% of respondents marked that a limited number of people in the respondent's work group used their database. This might suggest that novice designers are more likely to build databases addressing issues relevant to the tasks and people involved in their immediate surroundings.

- The respondents rating of their own appreciation of database design issues varied constantly between a little and very good. Whether this points to variable levels of conceptual understanding of design issues among novice designers or indicates that one cannot know how much one does not know cannot reliably be determined from the responses to this survey.

5.5 Conclusions drawn from the survey results

In terms of the three assertions described at the beginning of the chapter the survey results indicate confirmation of assertions one and two but are inconclusive concerning assertion three.

The results support the assertion that Σ database designers have a presence in the community. The evidence for this is that a significant percentage of respondents have provided answers that indicate they are consistent with the definition of a novice designer.

The low level of database deign assistance available to novice designers is born out by the results of the survey. The evidence supporting this assertion, while not as clear as that for the first assertion, is derived from a mix of methods used by respondents to design their databases. In addition to this the low incidence of respondents favourably reporting the helpfulness of the available resources for instruction concerning database design further reinforces this assertion.

The survey did not support the assertion that schemata designed by novice designers are unstable and difficult to modify without adversely affecting the existing functions. The evidence provided from the results does not repudiate this assertion either as it is too inconclusive to make a valid judgement on this issue.

The survey successfully identified ten respondents who were suitable and willing candidates to use as case studies to further examine the validity of assertion three. Case studies of four of these ten candidates are included in chapter 6.

SECTION FOUR

CASE STUDIES

6 Case studies

The novice designer survey revealed twenty-seven novice designers who indicated a willingness to participate further in this research. Of these, four have been chosen as case studies in the present report to support and reinforce the issues surrounding the use of relational DBMSs by novice designers. This section provides an overview of each case study and identifies and discusses issues of significance to novice designers born out in the case studies.

The four cases selected are chosen primarily because they appear from their questionnaire returns to be genuine novice designers, they were willing to offer their time to participate in interviews and they were within a two hour drive from the researcher's base making it feasible to interview them in their own environment and experience first hand the support they have and the databases they have built. Three of the four case studies are novice designers who responded to the articles placed in publications and each of these expressed a desire to have continuing involvement in this type of research. The fourth is a respondent from the mailing list who while willing to participate was prepared to commit no more than one hour to an interview and wished for no further part in the research.

Each interview consists of three sections. The first looks at the background surrounding the requirement to use a database to implement a solution. The second section queries the database built by the novice designer to identify characteristics that might be in common with other novice designer databases. The third section addresses questions concerning the future of the database and how the novice designer might address problems that may arise in the future.

Narratives of the four case studies used in this report follow here.

6.1 Case A. Socks and All

This case involves a part-time director of a hosiery manufacturing firm who requires a database that will act as an add-on module to the company's existing order and invoicing system to allow them to comply with their customer's systems.

The novice designer in case A is a semi-retired computer enthusiast who retains a part-time position as a director of the company. This is a small business employing between ten to twenty staff in a single location. Their principle business involves the production of quantities of hosiery lines to fulfil orders procured from numerous retailers throughout the country.

Manual systems are used to support business activities except for the general ledger and invoicing systems, both of which are computerised. These two systems, although not sharing functions nor data have been used successfully by the company's officers since the mid 1980s. The general ledger system, not the focus of this case study, is an off-the-shelf package that fulfils the company's requirements adequately. The invoicing system, developed off-site using Dataflex in 1984 and tailored to this company's requirements, is still used today to process most orders and produce invoices. Two years ago short-comings became apparent when a major customer required that invoices include stock unit (SKU) codes and bar-codes, features which were not catered for in the Dataflex system. It is at this point that the case's novice designer interceded.

Wishing neither to rewrite the existing system nor throw that system away, the case study's subject decided to build a supplemental application acting as an adjunct to the invoicing system catering for the unique requirements mentioned previously. The novice designer used Paradox version 3.5 to develop this application over a six month period working during evenings and spare time. Today, the finished application, ostensibly unchanged since its initial implementation, achieves the tasks for which it was built and allows more flexibility and functionality than the original Dataflex system. The interface between these two parts of the invoicing system is achieved manually by keying summaries shown on the Paradox reports into the Dataflex system.

The Dataflex system then generates an invoice top page that is attached to the detailed Paradox report pages and both are sent to the customer.

The novice designer in this case has a keen interest in information technology with a particular interest in CAD applications. He rates his computer skills as "good" and regularly uses word processing, DBMS, communications, spreadsheet, graphics and CAD packages. Having not had the opportunity to acquire any database concepts from database training courses he has relied on trail-and-error, product documentation and publications from third-party authors to build this package. Although he credits the manuals with guiding the initial schema design, the author's observation of irregular attribute distribution across tables strongly suggests a design driven primarily from the system's functional requirements. This suggestion was confirmed by the novice designer during the interview.

The Paradox system contains two central tables and approximately fifteen additional tables supporting the processes and functions required by the system. Most additional tables contain data either queried or copied from the central tables that therefore incur quantities of redundant and duplicated data in the database. The presence of this data causes no apparent inconsistent or unexpected results in reports or queries. Duplication of data is difficult to avoid when using Paradox as there is no facility to create and use views as exists in many other RDBMSs.

When posed the question of how to cope with a change in the system's data the novice designer's response was to build more tables handling the new requirements separately, and then interface manually with the existing systems. There were no strategies proposed for coping with a fundamental change to the data (change the data type of the SKU) within the existing system other than an appreciation of the drastic and pervasive impacts it would have on the current coding and processing within the system.

In this case study the designer has a desire to extend the scope of the system eventually replacing the Dataflex system. However he has become discouraged from doing so because of an apathetic attitude from other users of the system. There is also resistance from other users to any move away from the current system, particularly if it

means changing the interface with which they are familiar. He sees this proposed expansion of the system as an extensive task but is not daunted by the potential complexity of the enlarged database, rather he is frustrated by user apathy and resistance to change particularly being constrained to mimicking the existing interface that he rates as a poor design.

It is the author's opinion that the complexity inherent in expanding the Paradox system to incorporate the Dataflex system would impact adversely on the existing functions of the Paradox system and require considerable redesigning of the database schema. Consequently in this case there is potential for the effective use of the proposed schema re-design procedure.

In this case study the tables are in a rather de-normalised state and consequently require considerable re-organisation both at the logical and physical levels in order to move to a "cleaner" design. Unfortunately the application code is intrinsically linked to the data structures so unless there are compelling reasons to change the design, probably initiated from changes in system requirements, it is unlikely that it will be changed. However although a schema re-design procedure may not be used to effect changes in the physical database it would still be a useful procedure for refining and communicating the conceptual and logical database design.

6.2 Case B. Public Reservations

This case involves a public servant in local government who is building a database application to replace a manual system designed to manage the booking of halls and sports grounds in the local district.

The novice designer in case B is employed by a city council as a computer programmer, a position that seems contradictory to the requirements of a novice designer. However he has minimal formal qualifications and none that incorporate database concepts other than those gained by attending an in-house course on Paradox for Windows run by a local software supplier. Most of this novice designer's time is spent maintaining the council's legacy systems and coding new functions for these systems as required.

The Council, like most large organisations have considerable investment in computer systems and the infrastructure necessary to support them. Legacy computer systems cater for the data intensive applications associated with the council such as rating transactions, town planning, building permits and others. There are however many relatively trivial functions managed by council officers, one of which, the booking of public halls and sports grounds, is the subject of this case study. Until this novice designer's database application came on-line recently this function had been managed through a manual system by staff in one department. The impetus for moving this function onto a computerised system is unclear. There is a suggestion that from time to time that facilities are double booked or are reserved when in fact they are unavailable due to refurbishment or maintenance, however the novice designer did not regard the reasons behind the computerisation decision as relevant to his task of building the new system.

Equipped with knowledge gained from a recent Paradox for Windows course and the experience of previously designing two databases application systems using a different DBMSs the novice designer began building this application. Forms and procedures used in the manual system shaped the basis of the system's analysis. The design of the database schema then "fell intuitively" from the analysis work accomplished. The database consists of one main table and three or four subsidiary tables used for reporting and common queries. The main table does not appear to be normalised and contains redundant and derived data. The other tables also contain derived data and a considerable amount of data duplication. There has been some reports from users in the council regarding unusual results appearing in reports and queries, however because the application has only been in use for a couple of weeks the novice designer feels that these are minor teething problems and probably attributable to the application coding rather than the database design.

In response to the scenario of implementing a fundamental change in the system's data the novice designer's immediate reaction was to add columns or tables to the existing database, however on reflecting on the impact of such a change on the coding he decided that there would be little difference in the quantity of code to be modified between the strategies of adding tables and columns as opposed to modifying the

existing tables and columns. No changes of this nature had been required at the time of this interview.

While this system was being proposed there was some discussion, not formally stated as a system requirement, of linking the reservation database with the council's general ledger system. Design has continued without considering the implications of this linkage. There are now murmurs slowly growing louder of wishing to implement this linkage. The novice designer feels that if this becomes a formal requirement he will have to alter the database design to conform with aspects of the larger system. Although he feels quite competent to accomplish this it is something he realises may be fraught with unseen complications.

It is the author's opinion that the implications for this database application, if it is to incorporate linkages to the general ledger system, are that it will require extensive schema re-design and application re-coding to maintain adequate support for the existing functions. The novice designer concurs with this view and while apprehensive about the amount of work involved in re-coding is not concerned with how to restructure the database as the universe of discourse in this case is not complex. If this perception of low levels of complexity is accurate it seems likely that use of the proposed schema re-design procedure would be marginally effective.

In this case study the schema seems to require complete redesign both at the logical and physical levels to move to a more relational oriented design if the general ledger linkage is to be incorporated. Unfortunately the application code is closely linked to the existing database schema making much of this code unusable in an expanded system. Use of the proposed schema re-design procedure would be useful for designing, refining and communicating the conceptual and logical design of the new database scope encompassing the general ledger linkage function.

6.3 Case C Apples and Pears

This case involves a support officer in a research centre who has built a database to facilitate the allocation of orchard research blocks to scientists for both research and commercial projects.

The novice designer in case C is based at a regional research centre of a Crown Research Institute (CRI) and is employed as a computer support officer and local area network administrator for that centre. The centre employs between twenty and fifty people in positions including crop growers, orchard managers, scientists, laboratory assistants, administrators and computer support staff. The main objectives of this organisation are to effectively and efficiently manage resources to produce quality research and commercial results. Resources consist primarily of eight research orchards and several joint-venture commercial orchards.

The area of business that this novice designer's database is concerned with is tracking the use of blocks of orchards, or parts of those blocks, that are allocated to particular scientists for particular research experiments. This function has previously been undertaken by recording the data in a large and complex spreadsheet file. While the spreadsheet method had previously been adequate the increasing quantity of data being stored has highlighted problems that will be alleviated by the implementation of the novice designer's database. These problems include data entry errors either through typographical mistakes or an inability to locate the spreadsheet cell containing the correct item of data, requests from scientists and orchard managers for timely results to queries concerning the status and history of orchard resources, and the inability of the spreadsheet system to record required historical data.

To establish the schema for this database the novice designer turned to a mentor working at CRI headquarters who is familiar with the universe of discourse and experienced in the design and construction of database applications. This mentor has had significant involvement in the development and commercialisation of a software package offering a decision support system for the management of orchards (this system is hereafter referred to as Crop Manager). The case study's novice designer was persuaded that there are advantages in making the new system's data structure compatible with Crop Manager, in effect making it an additional module to Crop Manager. The suggested way to bring about this compatibility was to use the same key structures for tables in the new system as those existing in Crop Manager, a strategy that circumvented the need for a process to establish a suitable database schema. It also means that the database schema is tied to a design that may not best

suit it and the novice designer is required to incorporate any structural changes made to Crop Manager regardless of the impact on his system.

Data not stored in Crop Manager but required for the functions associated with this database have been incorporated as additional columns to Crop Manager's table structures or as look-up tables if addition is inappropriate. Consequently this application's database consists of two main tables both of which mirror the key structure of Crop Manager's tables and five or six look-up tables. There are also several other tables mirroring the key structure of tables found in Crop Manager but these exist because they are part of Crop Manager and they are not used for the functions currently supported by this system.

The case study's database currently stores client data supplied by managers of the eight orchards owned by the CRI. Data is supplied to the Orchard Resource Administrator by Orchard Managers in written format and entered into the database either by the Orchard Resource Administrator or an office typist. Problems associated with data entry are minimised by the use of on-screen forms that restrict and direct options available to the data entry operator. Access to the database is restricted at present to the novice designer, data entry staff and the Orchard Resource Administrator. Because validation of data entry is not deemed tight enough data entry access rights have not been granted to the Orchard Managers.

Paradox for Windows has been chosen for this database because the novice designer has had some exposure to the product through a university paper on horticultural management. Limited funding has been given to the project that means the most desired system objectives have been satisfied first with other features being added according to their relative priority as time and as funding becomes available. The highest priority features, which were supposed to be incorporated in the current version, are:

- Validation checking.

  Other than data restrictions imposed by Paradox's set of data types there is no validation done at the time of data entry. This increases the chance of data entry errors and precludes the possibility of assigning data entry access rights to the Orchard Managers. The novice designer states that incorporating this feature is

not technically difficult but is time consuming and consequently will not be done until funding becomes available.

- Incorporation of historical data.

  One of the major drawbacks associated with the use of a spreadsheet to record data in this universe of discourse is the inability to retain historical data. As each orchard resource is allocated to a new research experiment details of any previous experiment is over-written in the spreadsheet. Strategies for incorporating historical data into the system have yet to be considered by the novice designer, however he appreciates that enabling this function will require considerable application coding, additional tables and structural changes to some of the existing tables. To satisfy clients' expectations the incorporation of historical data needs to be part of this system. It is the author's opinion that existing functions, comprising of queries and reports, will need extensive re-working when this aspect of the system is incorporated.

- Central server installation.

  When the system is sufficiently robust and the Orchard Managers are granted the authority to enter and edit data there will be a requirement for the system to receive data from geographically dispersed locations. To cater for this it is desirable to install the system on a central server making it accessible to all interested parties. This option raises a number of additional problems such as managing concurrency and distribution neither of which have yet been considered by the novice designer.

In response to questions about the consistency and reliability of output from the current database the novice designer's belief is that most outputs are correct but he is not confident in the accuracy of some of the more complex queries. However he has no empirical data to objectively rate the system reliability as he has received negligible feedback from the Orchard Managers. The database does however contain a quantity of redundant data and derived data which may lead to inconsistencies.

As the database has so far only satisfied 70% - 80% of its initial objectives little thought has been given to the longevity of the system. It is envisioned that as features

are added the system's usefulness will increase and because it is viewed as a module to Crop Manager its life expectancy will tend to follow that product's life-cycle.

When asked to consider strategies for coping with fundamental changes to the data structure the novice designer suggested that should such an eventuality occur the CRI would release sufficient funding to allow him to effect a solution. Until such funding became available the novice designer thinks there is little point in considering how best to implement changes, his main concern in such a scenario is the amount of system down time and how this will impact on the system's clients.

Considering the issue of how experience with this database might affect the design of future databases the novice designer directed the author to a database design on the drawing board for the universe of discourse of managing survey data. The novice designer acknowledged that he has little idea of how to design a suitable database schema and currently has two non-normalised design proposals on a whiteboard, neither of which he considers to be adequate. He desires knowledge of conventions with which he can concisely depict and refine design ideas. He has heard of Entity Relationship Diagramming and considers this a suitable candidate to fulfil this need. When asked whether using sentences to clarify design issues would be of interest to him he felt they would be of more use when talking to clients about issues concerning the universe of discourse than when designing a schema whose universe of discourse is well known to him.

The database schema discussed in this case study inherits the consequences of being derived from an existing, largely unrelated, system. The proposed database redesign procedure could help in this case to establish a suitable schema to incorporate the yet-to-be-included features discussed. Because the novice designer has acquired much of the database schema from Crop Manager he is unaware of the reasons underlying the particular data structure. Consequently his familiarity with the structure is limited and his assessment of the impact caused to the system's functions by structural changes is likely to be inaccurate.

6.4 Case D Communications Breakdown

This case involves a communications technician who requires a database to track the maintenance of electrical and communications components and other equipment for which his organisation is responsible.

The novice designer in case D works for a nationwide company involved in the installation, maintenance and management of communications equipment. His regional office manages the equipment encompassed within an area radiating approximately one hundred and fifty kilometres from the centre where an office site located in a small central North Island town. The regional office staff consists of four technicians, an odd-job person, a secretary and a manager.

The company has a distributed computing platform and each regional office runs Microsoft software for most of their computing requirements. In addition to this the company has made a substantial investment in a mainframe relational database located in Wellington, available to regional offices by way of a nation-wide network. The mainframe database is used for financial and payroll operations and is administered by a third party consultancy firm. According to the novice designer there is widespread discontent throughout the company with regard to the quality of the services provided using this database but top management and the IT section are insistent that they attempt to realise benefits from the investment made in this DBMS and require that this product be the platform for any current and future systems development undertaken by the IT section.

The novice designer's manager saw a need for a system to efficiently track the status and maintenance history of modules (smallest identifiable pieces of equipment) falling within the jurisdiction of each regional office. When approached the IT section agreed that the idea had merit and proposed to incorporate an application comprising of those functions in the mainframe database 'sometime in the next few years.' The District Manager also saw the merit of the idea and championed the manager to find an alternate way of bringing the project to fruition. This is when the case study's novice designer became involved suggesting that they may be able to produce a database

application using the copy of Microsoft Access that they had recently received by mistake as part of new computer hardware and software supplied by the IT section. He received permission to pursue the project using Access within his regional office as his time and workload permitted. Eighteen months later he has an application that appears to satisfy many of the original requirements.

The database consists of fourteen tables, two or three of which are base tables, with the remainder being look-up tables. There appear to be no query tables or temporary tables in this design therefore reducing the prevalence of duplicated or redundant data. The novice designer has a strong conceptualisation of the database's schema to the extent of having a diagram depicting it residing on his office wall. The diagramming conventions used are reported as being of his invention, however a description of this is unavailable because the diagram has not been sighted by the author.

The process used to design the database's schema began with the novice designer's thorough knowledge of the domain. He then drew some flow charts to track data and used Access's sample database and forms' wizards to generate an initial version of the base tables needed. From speaking to his colleagues and allowing some time for mental reflection he made some changes to the initial design, mostly involving the addition or modification of some attributes. Having arrived at what he considered to be a suitable set of base tables he began to add look-up tables. Most of these were added during a short period, while the remaining few were added more recently as the requirement for them arose.

This database application development is considered successful to the extent that to-date it has proved to be a reliable and efficient way of tracking modules. The novice designer has a high level of confidence in the accuracy of the data in the database stating that in his opinion there is no duplicated, derived or redundant data present. A brief inspection of a few table structures, however, highlighted two instances of redundant and duplicated data. The first instance concerns the table "*Location*" where the two attributes "*Site_ID*" and "*Equip_ID*" are combined to produce a third attribute "*Location_ID*" that acts as the key field for the table. The second instance concerns a system generated bar-code number that is itself a concatenation of a number of numerical codes used as attributes in other tables in the database. In terms of

normalisation, the tables, although not consistently normalised throughout the database, are mostly at least in second normal form.

The system has been used successfully for some time in the novice designer's region. Three other regions have recently been asked to evaluate the application by using it to manage their module allocation processes. Feedback from these regions has been positive with a number of minor shortcomings highlighted and remedied. The novice designer feels that the database's usefulness is likely to increase in the short term future and intends to place a multi-user version onto the company's computer network so that modules can be managed on a nation-wide basis. The IT section still views this project as one that should be based on the mainframe database and consequently is not in favour of the networking idea. It seems, however that network security is lax to such point that anyone can install an application on the network and make it available to all staff. An additional expansion area suggested by interested parties is the incorporation of data reflecting the dollar value of modules. Although this facet of module management has never directly affected the novice designer he can see the reasons why such information is desirable for some management functions, and as his senior manager is championing the project he is quite acquiescent to the incorporation of this information. The reason these changes have not yet been incorporated is due to restrictions on time resources available to dedicate to these tasks.

When challenged with the scenario of coping with a fundamental change to the data the novice designer foresaw only a few minor problems. Upon reflection, however, he up-graded his impact assessment but still considered that any example proposed by the author could be isolated to a relatively small part of the database. He feels that managing changes in the functional requirements of the application such as the addition of data concerning the value of modules can be accomplished adequately by adding attributes to existing tables or perhaps creating new tables.

In the author's opinion the database in this case study could be modified to incorporate additional functions without adversely affecting existing functions. However the novice designer appears to have a simplistic view of how best to incorporate new meta-data to ensure that the new functionality can be achieved optimally. In this area the proposed re-design procedure is likely to be helpful to this novice designer.

In this case study the tables appear to be approximately normalised. They do, however contain a degree of redundant and duplicated data that has the potential to cause update anomalies and unreliable answers to queries although these have not been experienced to-date. The schema re-design procedure would likely be of assistance to the novice designer in the identification and elimination of these problems.

Due to the nature of the data in this database and the functions performed on that data there is limited application code attached to the database and little scope for the generation of derived data. Having a fairly clean separation between the application code and the database schema means that changes to the schema should only have a minor impact upon the code. However when the module value functions are included there is likely to be a need to generate some derived data. Decisions on how to effectively manage this derived data should be improved by better understanding the database schema, this improved understanding in turn could be facilitated by using the proposed schema re-design procedure.

6.5 Questions to be answered from the case studies

The questions in this section are important as they look at what strategies novice designers use to manage the issues of design and modification to their schemata. They show that the methods currently used have shortcomings and that novice designers need a structured mechanism to enable them to verify their schemata. This section contributes to the justification for a schema re-design tool such as that proposed in this thesis.

1. What are the problem solving strategies shown here?

Case A: Trial and error combined with some literature research are the main strategies exhibited in the case study. The novice designer has used both his knowledge and experience of software packages and third party reference books to familiarise himself with the concepts underlying the software and some of its capabilities. Having acquired this he has produced a number of tables that appear to address the objectives of the business problem. When problems and shortcomings become apparent he

iteratively modifies the design until it seems to address the situation adequately.

Case B: The primary method used in this case to design the database is the application of knowledge acquired from a DBMS product course. This method seems to consist mostly of collecting forms and reports pertaining to the current system and incorporating these into a computerised system. This would seem to have produced a database application fulfilling its required functions. However because the application has only been running for a short time there still exists some doubt as to its reliability with the users reporting unexpected system out-puts.

Case C: The design strategy shown in this case is in itself a type of evolution. The novice designer has taken some existing data structures, copied them, and amended them to suit his own purpose. Because it is desirable for this database to conform to the structure of Crop Manager the novice designer does not have an in-depth understanding of the data structures with which he is working. This lack of understanding will adversely affect his ability to evolve the database design as functional requirements change.

Case D: In this case study the process of schema design has been undertaken in a patient and methodical manner. The novice designer began with a clear understanding of the problem domain and then tried to identify the data structures by mapping the data flow throughout the universe of discourse. Once he was fairly confident that he knew what he wanted to do he set about doing it by adapting the sample tables provided in the DBMS of choice until they suited his purpose. There is also some degree of design by trial and error in this case as the design achieved by sample table modification continued to be revised as improvements were perceived.

2. Would a schema re-design procedure help (for evolving the current design).

Case A: It seems likely that a procedure enabling schema re-design would not be immediately helpful in this case. The way the database is applied to the company procedures is currently proving adequate to satisfy the organisational needs. Should the organisation want to update their systems in the future by integrating the functionality of the Dataflex system with the Paradox system then they may do this by extending the Paradox system. To do this it would seem likely that a procedure helping to evolve the current schema to capture additional data would be useful.

Case B: This database schema, being recently developed, is unlikely to benefit greatly from a schema re-design procedure. However after a period of time the proposed re-design procedure would be useful to enable the novice designer, whether it be the current person or someone else, to understand the intent behind the current schema and how it might be effectively integrated into a system incorporating the council's general ledger.

Case C: In this case it is likely that a schema re-design procedure would help in both understanding and evolving the current schema. The system is however constrained by the requirement to maintain data structure compatibility with Crop Manager. This effectively means that if it is desirable to extend the scope of the current system the designer will not have the latitude to revise a significant proportion of the current schema. Therefore any evolutions of this database schema is going to contain a significant number of design compromises. Despite this it is still desirable to know what an uncompromised design would look like so that the reasons for, and implications of, the design compromises can be assessed. This knowledge will help the novice designer choose a set of design modifications that suit the new functions yet maintain compatibility with the parent system.

Case D: It seems likely that a schema re-design procedure would be useful in this case as the database design has undergone quite a lengthy evolution from its initial design stage. It seems there is a strong possibility that the application will be expanded in the near future and before this happens it would be prudent to examine the existing schema with a view to reducing the number of potential design problems. There are also key structure problems in a number of tables that might be highlighted by the application of a re-design procedure.

3. Does this person have a clear enough picture of the current structure to be able to modify the structure in such a way that the existing functions are maintained and new functions can be optimally achieved?

Case A: The novice designer in this case appears to have a thorough understanding of the current database schema however is very unsure of how he might incorporate this structure into a database whose scope incorporates the functions currently managed by the Dataflex system. The novice designer stated himself that although he is very familiar with the various tables in the current structure but has no strategy available to him to modify the structure to achieve new functions while maintaining existing ones.

Case B: The novice designer in this case is confident he has a very clear understanding of the database schema. Because the current structure is perceived as simple and the integration with the general ledger system is seen as a non-complex task the novice designer is confident of his ability to incorporate this function into the current database design. However because the design has been undertaken without consideration of the structures required for the general ledger system it is likely that there will be complications, unforseen by the novice designer, that will make this a non-trivial task.

Case C: In this case it seems that the novice designer has little understanding of the data structures in the database save for those few tables that his system centres around. The constraints imposed by having to maintain

compatibility with Crop Manager's data structures poses an additional set of problems should the database be required to widen its scope. The most likely strategy to use for incorporating additional functionality into this database is to add tables and attributes. However use of this strategy will incur a constantly increasing risk of data integrity problems.

Case D: The novice designer in this case has a good understanding of the data structures in the current schema. His problem arises from an inability to be able to judge the quality of the existing structure. The design has a few fundamental problems that if they were carried into a database with an expanded scope would cause some complications that would adversely impact on the performance and reliability of the database.

4. If this person does not have a clear understanding of the current structure or cannot modify the structure for new functions, how will a database re-design procedure help in this situation?

Case A: The requirements of this situation are such that the current database schema fulfils a relatively small proportion of the potential functionality. Consequently in this case it might be desirable to design a complete application from scratch rather than evolving the existing schema. If this course of action were taken then the schema re-design procedure would be of minimal use. It could however still be used to clarify the data structures in the current database that could then be incorporated in the new system thus saving some conceptual design work.

Case B: Other than to communicate the intention behind the current design the proposed re-design procedure offers little to alleviate any difficulties in this situation. If the database was older and had undergone evolutionary changes the re-design procedure would be of more utility in this case.

Case C: The database re-design procedure will allow the novice designer to construct a version of the design without the compromises present for both physical and compatibility reasons. This will allow the novice designer to see how any new functions required should be incorporated into the

database. Following the reasons for compromises in the existing structure should assist in identifying where modifications should be made to the existing schema.

Case D: In this case a re-design procedure would help identify problems in the database structure rather than enlighten the novice designer to aspects of the universe of discourse. The procedure should be able to improve the design and therefore allow the novice designer to place more confidence in the quality of the database's design.

6.6 Results obtained

All the case study interviews varied from the interview frame-work depicted in appendix three. Each interview ran its own course because of the different environments the interviewees' work in and the different universes of discourse with which each is involved. Use of the framework ensured that as far as practical the same issues were raised with each interviewee which allowed some comparability between case studies to identify common themes and differences. Issues expressed by only one interviewee and not necessarily applicable to the others are also expressed here.

In the first section the subjects gave a number of reasons why they chose to develop their own database. The common themes running through these responses are that there were no other practical ways open to them to get the database developed and that they considered it advantageous to use their knowledge of the universe of discourse in the development. They seemed to have little problem defining the scope of the database attributing this in the main to their intimate knowledge of the functional requirements of the business and a strong working relationship with other workers. None of the case study subjects used forms and reports as design aids believing again that their knowledge was sufficient. There is a common theme present in all the cases concerning the impetus for the project, which indicates that it came from a dissatisfaction with the efficiency and reliability of the systems currently supporting the situation.

The second section revealed agreement between the novice designers on several issues. All four felt that they had a clear enough mental picture of the database structure to be able to describe it to the researcher. Three reported that the number of tables in the database had not increased since its initial design although they all felt that more tables were likely if the scope were to change in the future.

The quality of the database in terms of redundant, derived and duplicated data was a question that the subjects had trouble in answering. From the brief look at each database that the author had, it seemed that all four had problems to some degree in each of these areas. All novice designers however felt that their databases were reliable enough to satisfy the requirements for which they were built.

The long term usefulness of the systems is a subject about which all cases study subjects felt confident. They felt that even if the databases were to stagnate at their present scope they would satisfy the business requirements for the foreseeable future. However after discussing the potential inherent in extending the databases' scope all respondents except one felt that they should extend their database. Extending the databases is likely to make them more useful and important business tools in the future.

Coping with changing the characteristics of fundamental data tended not to concern the novice designers until the author explained how such changes might impact on their system. At this point all four revised their estimations and began to become concerned about the problems with which they might be faced.

In the third section all subjects considered that the database achieves the purpose for which it was built. Three of them are keen to implement changes each involving extending the scope of the application as discussed earlier. They have not implemented changes before because of a mixture of funding restrictions, time constraints and a lack of database knowledge and concepts required to implement them with confidence.

Looking to the future all four indicate that when data requirements or functions in their universe of discourse change they will cater for these changes by adapting the current structure. In the main they would be reluctant to modify or delete any of the existing structures and would instead prefer to add tables and attributes where they felt necessary. They gave little thought to how they might assess the impact of changes or

additions to the code and functionality of the system opting for a trial and error approach in which they will address problems as they uncover them.

When discussing formal database design aids all of the novice designers felt that any tool be it diagrammatical or sets of sentences would be a vast improvement on their experience of database design. Even after designing what seemed to be fairly successful databases they do not feel confident in designing other databases or extending the scope of the ones they already have.

6.7 Summary

In general the findings from the case studies tended to support the assertion that a sample of novice designers are likely to encounter similar types of problems when using DBMSs to design and build systems. They have a number of strategies, such as trial and error and iteration, that can be used when deciding what tables to place in the schema. They tend to use a mixture of these strategies as they see appropriate to arrive at what seems a suitable design. However when they are presented with the challenge of modifying their design, they have few strategies available to them and no way to judge the quality of the final schema.

SECTION FIVE

REDESIGN PROCESS

7 Schema re-design process (SRDP)

This chapter describes a strategy aimed at providing a structured means to migrate database schemata such as those discussed in the case studies to more normalised designs. The predictability of the process described makes it a likely candidate for incorporation as a component of a CASE tool or to be implemented as a stand alone piece of software. Although software production is outside the scope of this thesis such an eventuality would make it possible for novice designers with existing problematical database schemata to migrate their designs without the involvement of a professional designer.

The rationale for the design of the of the schema re-design process specified in this section is derived from an explanation of the optimal normal form algorithm (ONF) of Nijssen & Halpin (1989, pp 254-279) and as presented in chapter four of this thesis. The complete database schema re-design process involves seven stages as shown in Table 3. This chapter concentrates on the first five of these stages. These five stages begin with an existing database schema and conclude with the incorporation of semantic constraints external to the existing database being included in a new NIAM type diagram.

Stage 1 - Transformation of existing schema to populated diagram with standard predicates.
Stage 2 - Addition of meta-data from the database to explicitly show constraints.
Stage 3 - Creation of example sentences.
Stage 4 - Verification of example sentences by universe of discourse expert.
Stage 5 - Incorporation of additional semantic constraints.
Stage 6 - Completeness check and generation of new logical model (NIAM type diagram)
Stage 7 - Generation of new physical schema (relational).

Table 3. Seven Stage Re-Design Process.

To explain the procedures used throughout these stages two examples are presented here. The first example comes from a database application developed under contract

by the author and a colleague to manage administration aspects for a commercial venture. This example is used to demonstrate different concepts involved in the reverse-design process, some possible consequences arising from the process and how these might be managed. The example schema is not complete as only selections of the database design required to demonstrate the relevant issues are presented. Results are discussed in the light of a comparison with results obtained from colleagues concurrently designing a database schema for the same universe of discourse utilising a conventional implementation of the NIAM methodology.

The second example is taken from the fourth case study, Communications Breakdown, contained in this report. This database is designed by a bona fide novice designer and is included in the present report in its entirety. The first five stages are documented in detail with the remaining stages described more briefly as the concepts underlying these stages fall outside the main focus of the present research.

7.1 Stage one

The process for stage one involves the translation of constructs explicitly recorded in the database's meta-data (referred to as group A constructs) from the existing database into a NIAM type diagram as shown in Table 4. The table construct in a relational schema is not directly translated into a NIAM type diagram. However for exposition purposes it is desirable to render the relational schema by translating one or two tables per page of the NIAM type diagram. As shown in the examples that follow later in this chapter this method offers a simple mapping of the relational schema into a NIAM type diagram and leaves little chance of missing or duplicating any part of the relational schema during translation.

DBMS Construct	*NIAM type diagram*
Table	Not Explicitly Represented
Attribute	Logical Object Type
Data type	Not Diagrammatically Represented
Primary Key	Uniqueness Constraint
Concatenated Key	Objectified Fact

Table 4. Group A constructs.

A table's attributes are mapped into a NIAM type diagram as logical object types involved in fact types. Each attribute is involved in a fact type with the logical object

type representing the primary key attribute or attributes. If they are involved in fact types of arity greater than one then they are explicitly represented on the diagram as ellipses otherwise they are depicted by roles in the fact types to which they belong. Logical object types can either be lexical object types (LOT) or non-lexical object types (NLOT) the latter are often values and are represented by dashed ellipses. One complication arising from mapping each table into a separate diagram is the forced replication of logical object types across different diagrams. This problem is controlled by depicting these replications as doubled ellipses as shown in Figure 31.

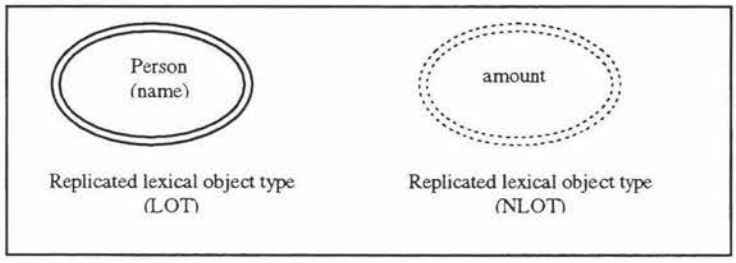


Figure 31. Replicated object type representation.

Data types are not represented on the NIAM type diagram but should be retained as part of the definition of logical object types. Primary key structures are the other main component extracted from the relational schema at this stage. If the primary key consists of a single attribute this is translated into a single logical object type. Any fact type that this logical object type is involved in, has a uniqueness constraint on the role immediately adjacent to it. In accordance with step four of the CSDP (Nijssen & Halpin, 1989) this means that all associated fact types on the diagram for this table's diagram will either be of arity one or two.

Primary key structures consisting of two or more attributes are mapped to an objectified fact type with an arity of the same length as the number of attributes involved in the key. The uniqueness constraint for this objectified fact type will extend across all of it constituent roles. Any fact types involving the objectified fact type will again be binary or unary fact types whose uniqueness constraint will also be placed on the role adjacent to the objectified fact type.

7.2 Stage two

Stage two involves transferring additional constraints and semantic information held in the DBMS into the NIAM type diagram. This information is typically more difficult to extract from the database schema than group A constructs and is likely to be supported to varying extents by different DBMSs. Table 5 shows a set of semantic information constructs potentially supported by relational DBMSs possibly used by novice designers.

DBMS Semantic Construct	*NIAM Type Representation*
Foreign Key	Fact Type Relationship
Table look-up	Not explicitly depicted
Minimum Value	[>n]
Maximum Value	[<n]
Restricted Set of Values	[A,B,C]
Range of Permitted Values	[n...n_2]
Explicit Mandatory Attribute	Solid circle attached to object
Default Attribute Value	Default "A"

Table 5. Group B Constructs.

Foreign keys are depicted by a LOT involved in more than one fact type, involved in a fact type with a second LOT involved in more than one fact type. This indicates the presence of a foreign key relationship between two LOTs. The nature and direction of the foreign key relationship is indicated by the common fact type's uniqueness constraint.

If the uniqueness constraint exists over a single role in the fact type then migration of the foreign key attribute goes from the LOT attached to the non-unique role to the LOT attached to the unique role.

If the fact type is of higher than binary arity then the migration of the foreign key attribute goes from the LOT attached to the non-unique role to the LOT attached to the unique roles.

Most relational DBMSs possess some capacity for the designer to add semantic constraints to the schema that are not inherent in the basic schema design. The form of these capabilities varies from DBMS to DBMS in both the extent of supported options and the method in which each semantic constraint is implemented by the designer. The selection shown in Table 5 is drawn from Borland Paradox 4.5 DBMS. Other

products' constraint options fulfil the same objectives but in doing so employ subtly different methods.

Most of these constraints can be explicitly represented on a NIAM type diagram by annotating text adjacent to the relevant object type. The exceptions are the table look-up function and the mandatory / optional function. The table look-up function requires that data entered into the object table first exist in the table that possesses that attribute as a primary key. This constraint is in effect the rule of referential integrity (Date, 1986b) and is implicitly present on a NIAM type diagram by a replicated object acting as a primary key object in another subset of the diagram. The mandatory nature of an object in a fact type is explicitly depicted by attaching a solid circle to that object. This is shown in Figure 32 where it is mandatory that each patient must have a gender and optional whether patients do or do not have phone numbers[12].

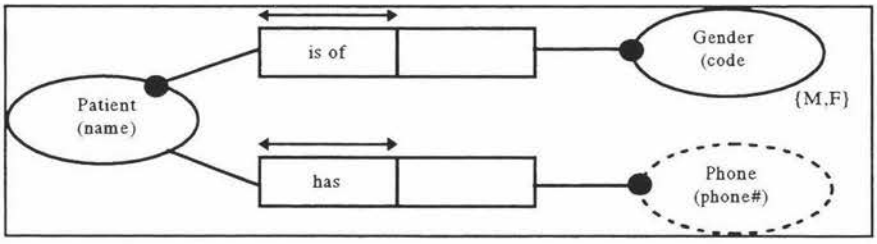


Figure 32. Depiction of Mandatory and Optional Constraints.

7.3 Stages three and four

Stage three generates a series of example sentences that are subsequently validated by the novice designer in stage four. These sentences extract data from the existing database schema and incorporate it into fact types to create instances of fact types. Although a series of these sentences can be included on the NIAM type diagram it is usually more appropriate to list them in a separate report. The procedure for generating a representative set of example sentences is shown in Figure 33.

[12]The mandatory role depictions on the gender and phone objects are shown for exposition purposes and would often be omitted from a NIAM type diagram as their function is implicit.

1. Take a value for each role from the data and incorporate them into the first sentence.
2. For the second sentence: change the right most role value whilst keeping the others the same.
3. To produce the next sentence: return the value of the right most role to what it was in the first sentence and change the value of the role to its immediate left whilst keeping all others the same.
4. To complete the sentence set: keep returning the value from the previous change and adjust the next role to the left.
5. When the value of the role on the far left has been adjusted a complete set of example sentence has been produced.

Figure 33. Example Sentence Generation Procedure (derived from InfoModeler).

From the set of example sentences in Table 6 it is possible to determine the correct uniqueness constraints provided sufficient expert knowledge of the universe of discourse is available. Assuming sentence 1 by itself is valid then if sentence 2 is invalid while all others are valid a uniqueness constraint should be attached to the left most role. If sentence 3 is invalid with all others being valid then a uniqueness constraint should be attached to the second role. If sentences 2 and 3 are both invalid whilst sentence one is valid then uniqueness constraints should be placed individually on both roles. Finally if all sentences are valid then the uniqueness constraint should be placed across all roles.

Figure 34 and Table 6 show how data may be included in a fact type and how that same data might be presented for validation purposes.

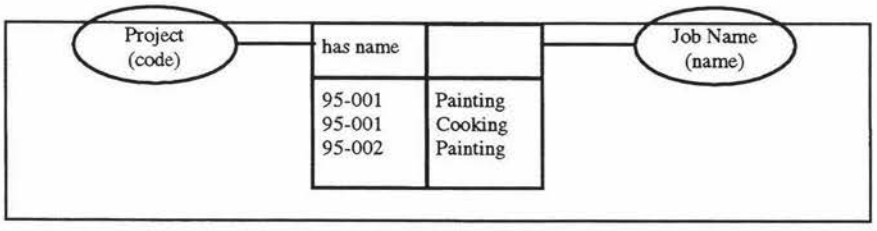


Figure 34. Example Instances for Binary Fact Type.

The *project* with code **95-001** has name of *job name* **Painting**
The *project* with code **95-001** has name of *job name* **Cooking**
The *project* with code **95-002** has name of *job name* **Painting**

Table 6. A Representative Set of Example Sentences.

The number of example sentences that comprise a sufficient quantity to validate the uniqueness constraint conditions depends on the arity of the fact type. If it is a unary fact type then two example sentences are sufficient. Binary facts types such as

Figure 34 require the generation of three example sentences as shown in Table 6 and ternary fact types as shown in Figure 35 require a set of four example sentences shown Table 7.

In general terms, where n = the arity of the fact type, non-reducible fact types require the generation of n+1 example sentences in order to generate a set of example sentences that can be used to validate their uniqueness constraints.

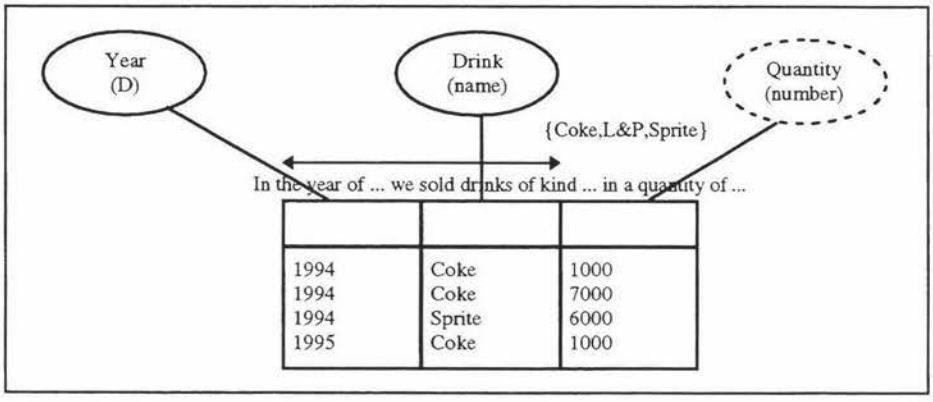


Figure 35. Fact instances Included in Diagram.

In the *year* of **1994** we sold *drink* of kind **Coke** in a *quantity* of **1000.**
In the *year* of **1994** we sold *drink* of kind **Coke** in a *quantity* of **7000.**
In the *year* of **1994** we sold *drink* of kind **Sprite** in a *quantity* of **1000.**
In the *year* of **1995** we sold *drink* of kind **Coke** in a *quantity* of **1000.**

Table 7. Population of Example Sentences.

When validating fact types with higher arity than 2 it is possible that validation of the set of example sentences will indicate the application of an illegal constraint pattern. In these circumstances it is likely that the fact type in question can be reduced into constituent parts without losing any semantic information. This reduction should be done, with re-generated sets of example sentences returned to the universe of discourse expert for validation. This cycle should be repeated until all fact type have been validated.

7.4 Stage five

Stage five incorporates further semantic information from the universe of discourse expert into the NIAM type diagram. This information may include constraints such as occurrence-frequency constraints, equality constraints, sub-type constraints, sub-set constraints and exclusion constraints. These are shown in Table 8 with their respective diagram notations described.

Constraint	*Representation*
Occurrence frequency	Maximum frequency number placed adjacent to role
Semantic Mandatory Attribute	Solid circle attached to object
Equality	Doubled arrowed dashed line between participant roles
Sub-type	Arrowed line between object types
Sub-set	Single arrowed dashed line
Exclusion	Dashed line with an X between participant roles

Table 8. Group C Constructs.

These constraints are discussed in the present report during the descriptions of stages six and eight of the CSDP and in chapters six and eight of Nijssen & Halpin (1989). They are not explained in detail here as their applicability to problems encountered by novice designers is likely to be limited. With the exceptions of mandatory attributes and occurrence frequency constraints there is no support for them from the current versions of the main relational DBMSs most likely to be used by novice designers. The value derived from their inclusion on the NIAM type diagram is the additional understanding they convey about the universe of discourse. Although they are seldom mapped directly into relational schemata they are often incorporated into application code built on top of the schema. They also provide a rich source of documentation available to interested parties should the original database designer be unavailable. Mandatory attributes are included in both group A and group B because they can be recorded in the RDBMS or can be derived semantically from the universe of discourse.

7.5 Stage six

As with step nine of the CSDP stage 6 of the SRDP entails a completeness check of the NIAM type diagram while generating a new NIAM type diagram logical model. Firstly the NIAM type diagram produced at the end of stage five undergoes a population check to see that the revised fact types represent the revised sets of example sentences. Next a check is performed to ensure there is no redundancy of fact

types in the diagram. Finally a check is performed by the expert in the universe of discourse to confirm that all aspects of the universe of discourse are represented in the diagram. Each of these checks is described earlier in step nine of the CSDP in the present report and in chapter nine of Nijssen & Halpin (1989).

7.6 Stage seven

The final stage of the SRDP is the generation of a new physical schema. This is done by application of Nijssen & Halpin's (1989) optimal normal form algorithm as described earlier in the present report. The schema produced from the SRDP will be highly normalised and is likely to require further refinement before implementation. These refinements will usually be de-normalisation decisions undertaken for performance reasons. Heuristics and guidelines concerning de-normalisation decisions are largely dependent on the particular DBMS and database application and are outside the scope of the present report.

Each of the stages in the SRDP allows for iterative cycles where, in order to produce a better result, it may be necessary to return to a previous stage before progressing to the next. The volume of data managed in the re-design process and the methods of managing that data suggest that the incorporation of the SRDP should be a useful feature added to a CASE tool such as Infomodeler.

Infomodeler is a CASE tool based on the NIAM methodology. It supports many of the constructs and constraints described in the CSDP. The following chapter contains a number of NIAM type diagrams drawn using Infomodeler, however some have been manually annotated to explicitly incorporate some common DBMS schema constructs. Infomodeler has primarily been used in the present report for its diagramming features. Like most CASE tools it retains considerable information concerning the universe of discourse that it uses in its detailed reports and transformations rather than in its diagrammatical representations.

8 The Administration Database Example

This chapter describes an application of the SRDP to a candidate database whose schema design is well known to the author. Its purpose is to show the method in use and typical results it may generate. The administration database schema is used in this chapter because it possesses sufficient issues to highlight many of the constructs inherent in the SRDP.

The administration database has undergone several owner-initiated evolutions during its two year history. The original need for a system arose from the failure of the existing administration system and the non-existence of off-the-shelf packages to adequately meet the owner's requirements. The system's original scope consisted of attributing staff hours and associated costs to client sponsored projects undertaken by the owners. In addition an invoicing capability was required along with significant reporting on many aspects of administering the owner's organisation.

The DBMS platform used to build this system is Paradox for DOS version 4.5. The original database schema consisted of six base tables containing the most volatile data, eight secondary tables acting as look-up tables or containing relatively stable data, and approximately twenty temporary or query tables. The reason for the existence of a high quantity of incidental tables is that Paradox does not support the Relational concept of views, instead it requires the database designer to create and manage persistent query tables.

The first evolution of the schema potentially impacting on the existing functions came about because the owner wished to integrate the general ledger function into the system. To incorporate this requirement additional attributes needed to be added to existing tables. Without being able to use views, isolating the impact of modifications to the database schema from the application code is a difficult but important task. Consequently evolutionary schema modifications tended to concentrate on minimal impact on the code at the expense of an optimally normalised design.

The second evolution involved the addition of a number of functions including security functions and more comprehensive management of the proposal and project functions. It is the database schema after these modifications that is used in this example. It constitutes a database schema originally normalised but now in an untidy state of design and offers a suitable subject to demonstrate strategies for managing many design characteristics potentially found in Σ databases.

8.1 The original database schema

The database schema currently consists of 47 tables, 19 of which are base tables[13]. The following seven tables contain examples of the main design constructs under discussion here: Account, Credits, Project Expense, Staff, Timespent, Task and Account Budget.

Using Paradox's table structure report, the following information has been extracted concerning each of the seven tables. The field name of each column, the data type and length of each attribute, and which attributes contribute to the table's primary key. Data types available in Paradox include alpha-numeric (A), small integer (S), number (N), date (D), currency ($) and memo (M). This information is rendered using NIAM type diagrams.

8.1.1 Stage 1: Transformation of existing schema to populated diagram with standard predicates

Table 9 shows the Group A constructs obtained from Paradox for the *Account* table and the format in which they are obtained.

The NIAM type diagram depiction of the *Account* table is shown in Figure 36. This table has a concatenated primary key composed of *Account number* and *Sub account number*. Following the NIAM reverse engineering algorithm the key becomes the central object in the diagram, in this case it is an objectification of the fact type

[13]These tables contain the majority of the data from the universe of discourse that is not stored primarily for processing purposes nor for efficiency considerations.

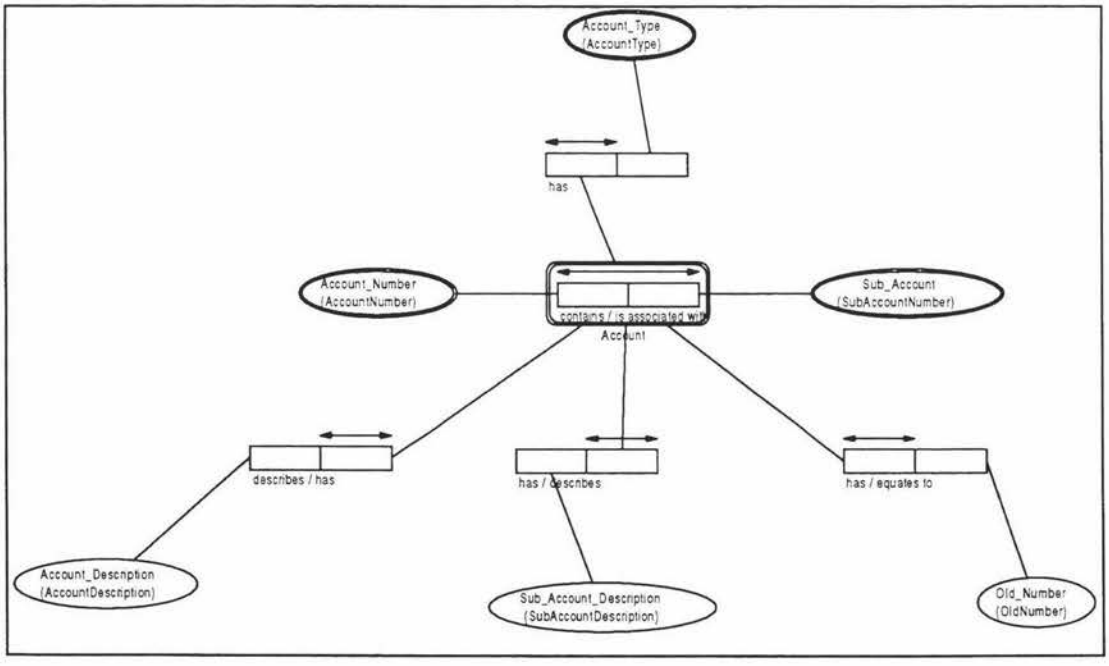
between *Account number* and *Sub account number*. Instead of naming this object as a concatenation of its associated object names it inherits the name of the original table, in this case *Account*.

Table 9. Account Table

Field Name	*Field Type*
Account number	A5*
Sub account number	A5*
Account description	A30*
Sub-account description	A12*
Old number	A45
Account type	A8

To aid readability and avoid cluttering, the diagram in Figure 36 represents a single logical portion of the database schema. Objects bordered by a double line represent those objects that appear in other areas of the schema and therefore also appear on other diagrams.

Figure 36. Account Table NIAM type diagram.



The double headed arrow indicates a uniqueness constraint that is implicit in the schema. It translates to one *Account* has one and only one *Account_Type* and one *Account_Type* can classify many *Accounts*. For the objectified fact type *Account* the uniqueness constraint translates to "one *Account number* contains zero or more *Sub account numbers*" and "one *Sub account number* is associated with zero or more *Account numbers*."

Field Name	Field Type
General Ledger number	S*
Account	A5*
Sub Account	A5*
Invoice number	N
Project id	A8
Credit note sig	A6
Invoice date	D
Client id	N
Dept	S
Invoice amount	$
Month	S
Year	S

Table 10. Credits Table.

In Table 10, which shows the Group A constructs obtained from a Paradox *Credits* table, there is a three part key which is rendered in the NIAM type diagram shown in Figure 37 as an objectification of the fact type involving the *General ledger* object type and the previously objectified *Account* object type.

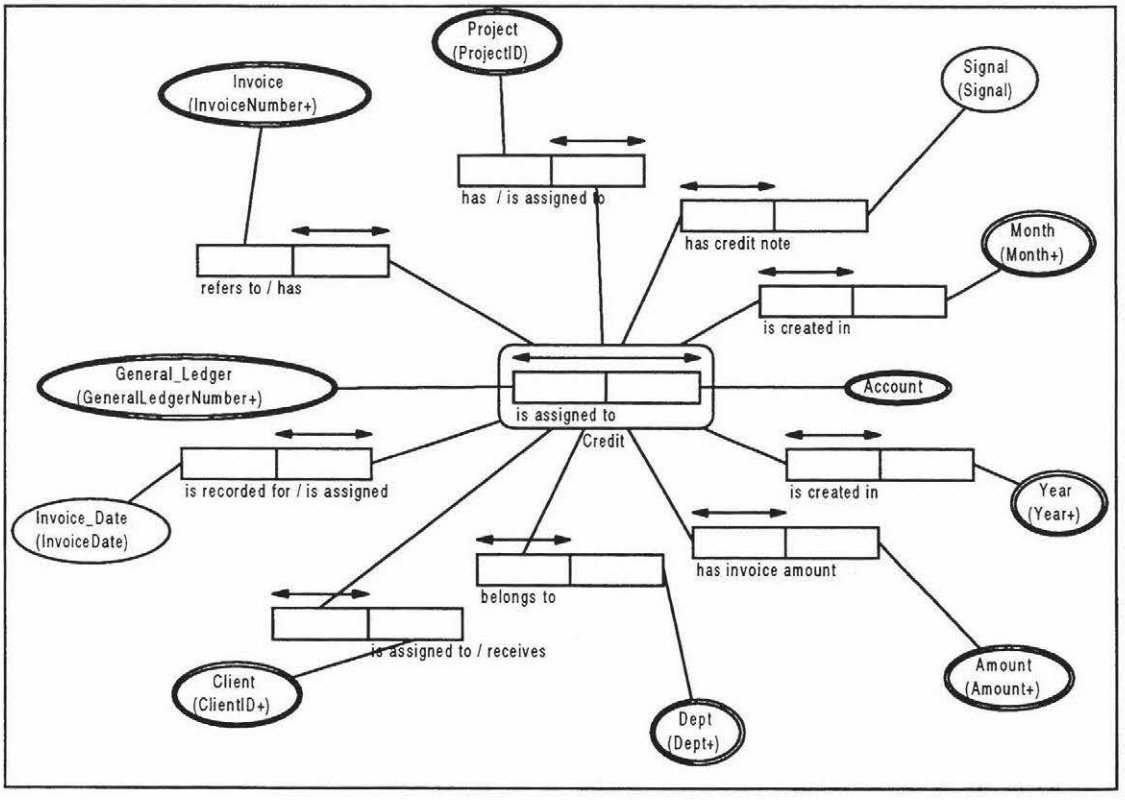


Figure 37. Credits Table NIAM type diagram

Most objects types in Figure 37 are linked to object types found on diagrams representing other tables of the schema.

Field Name	*Field Type*
Project id	A8*
Expense code	S*
Date	D*
User desc	A15*
General ledger no	N
Expense amount	$
Chargeable signal	A1
Week ending date	D

Table 11. Project Expense Table.

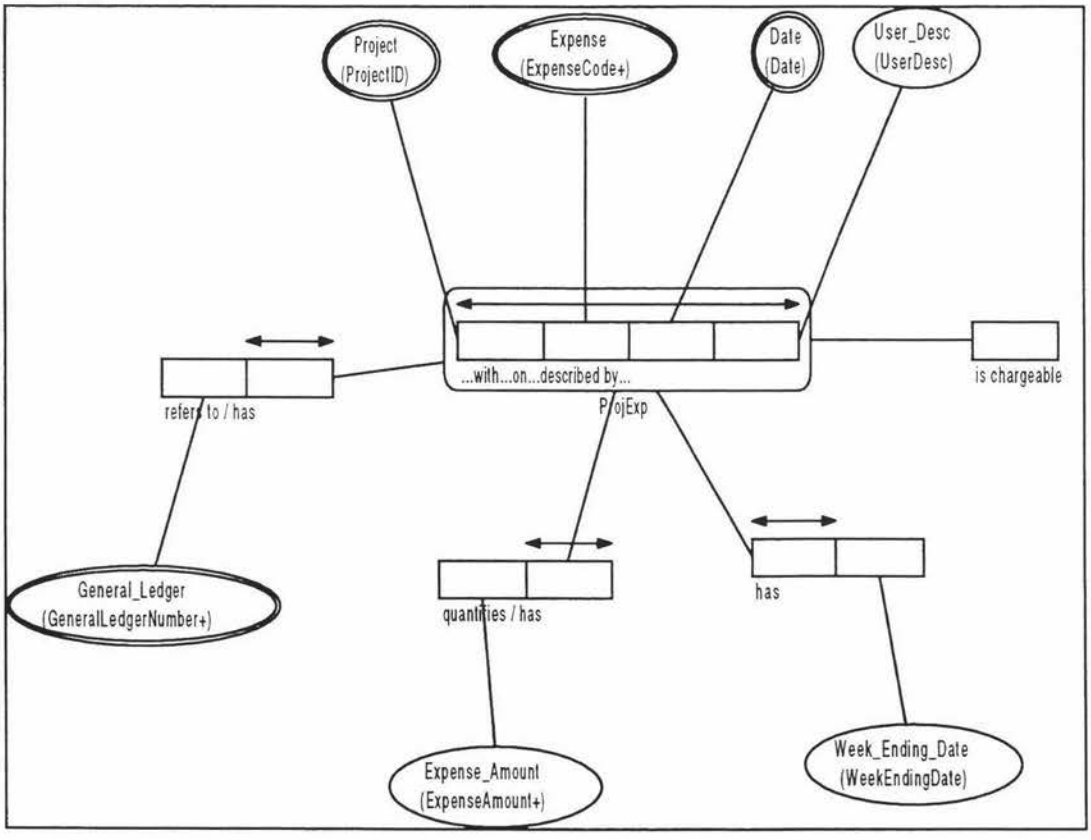


Figure 38. Project Expenses Table NIAM type diagram

Table 11 and Figure 38 show the Projects Expense fields and contain two constructs not exemplified previously. Firstly there is an objectified quaternary fact type (*ProjExp*) between the object types *Project*, *Expense*, *Date* and *User desc* that combined constitute the key for the table. Secondly there is a unary fact type labelled

is chargeable that is the translated form of the logical attribute chargeable signal, meaning the table row is either chargeable or is not chargeable.

Field Name	*Field Type*
Project id	A8*
Project name	A25
Start date	D
Expect end date	D
Agreed budget amount	$
Deposit amount	$
Client id	N
Income received	$
Project leader	N
Inception date	D
Proposal sent date	D
Proposal follow up date	D
Proposal discontinue date	D
Group code	N
Project actual end date	D
Overhead amount	$
Project acceptance	A1
Monthly accrual amount	$
Project duration	S

Table 12. Project Table.

The version of the Project table shown in Table 12 and Figure 39 is significantly different from the one contained in the original implementation of this example schema. It incorporates the meta-data from two logical areas of the universe of discourse. The original version was concerned only with project information but later an additional need arose to record information about proposals, this additional information is incorporated in the diagram. This is a busy diagram but avoids becoming cluttered by allowing several fact types to be connected to one representation of several semantically equivalent objects.

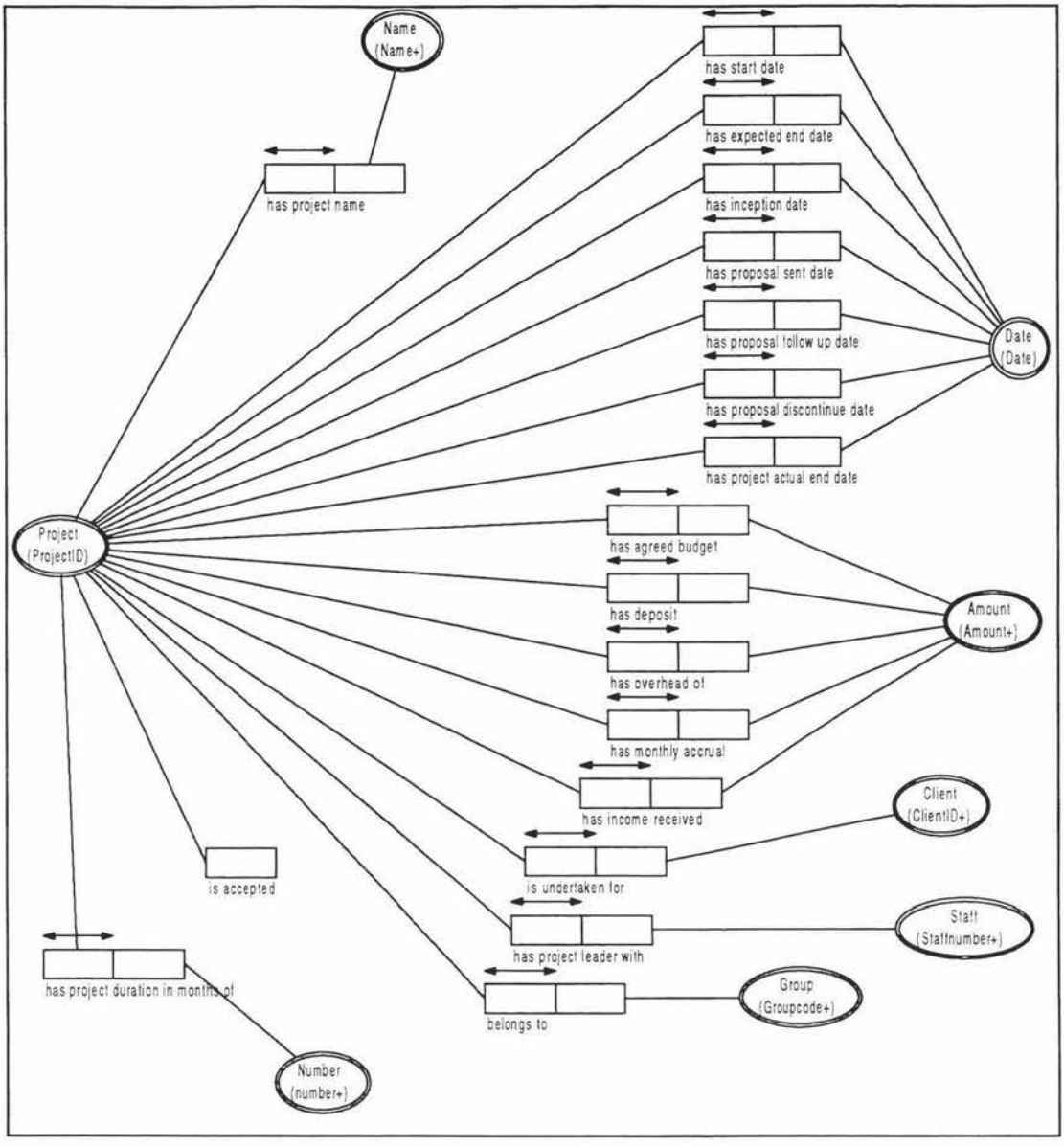


Figure 39. Project Table NIAM type diagram.

8.1.2 Stage 2: Addition of meta-data from the database to explicitly show constraints.

Each of the tables described in the previous section has additional constraints, termed in the present report group B constraints, that are stored in the database. These constraints are added to the NIAM type diagrams in this section. The constraints present in these tables represent only a portion of those supported by Paradox which in turn are only a portion of possible Group B constructs in other DBMSs.

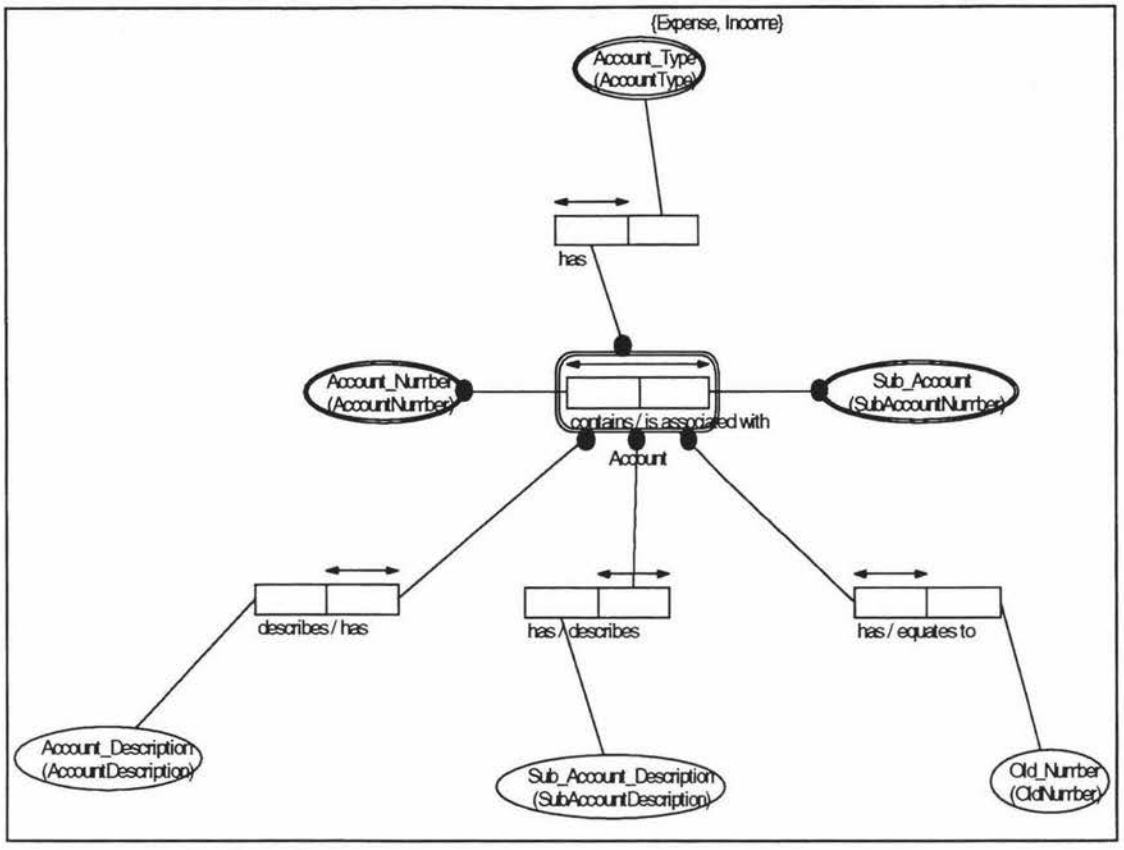


Figure 40. Account NIAM type diagram with Group B Constructs.

The depiction of mandatory roles on the two object types contributing to the objectified fact type *Account,* and the *Account* object type's role in the other four fact types remaining are the most dominant group B constructs shown in Figure 40. An additional constraint is shown on the *Account_Type* object type that restricts it to two possible values; "Income" and "Expense."

The diagram for the Credits table shown in Figure 41 again shows mandatory roles for the object types participating in the objectified fact type named *Credit.* Mandatory roles are also depicted between *Credit* and the six object types that have been recorded as mandatory in the DBMS schema. In addition the object type *Signal* has a set of two possible values; CREDIT and DEBIT. The object type *Month* has a range of possible values between 1 and 12 inclusive and the value of the *Year* must be in excess of 1994.

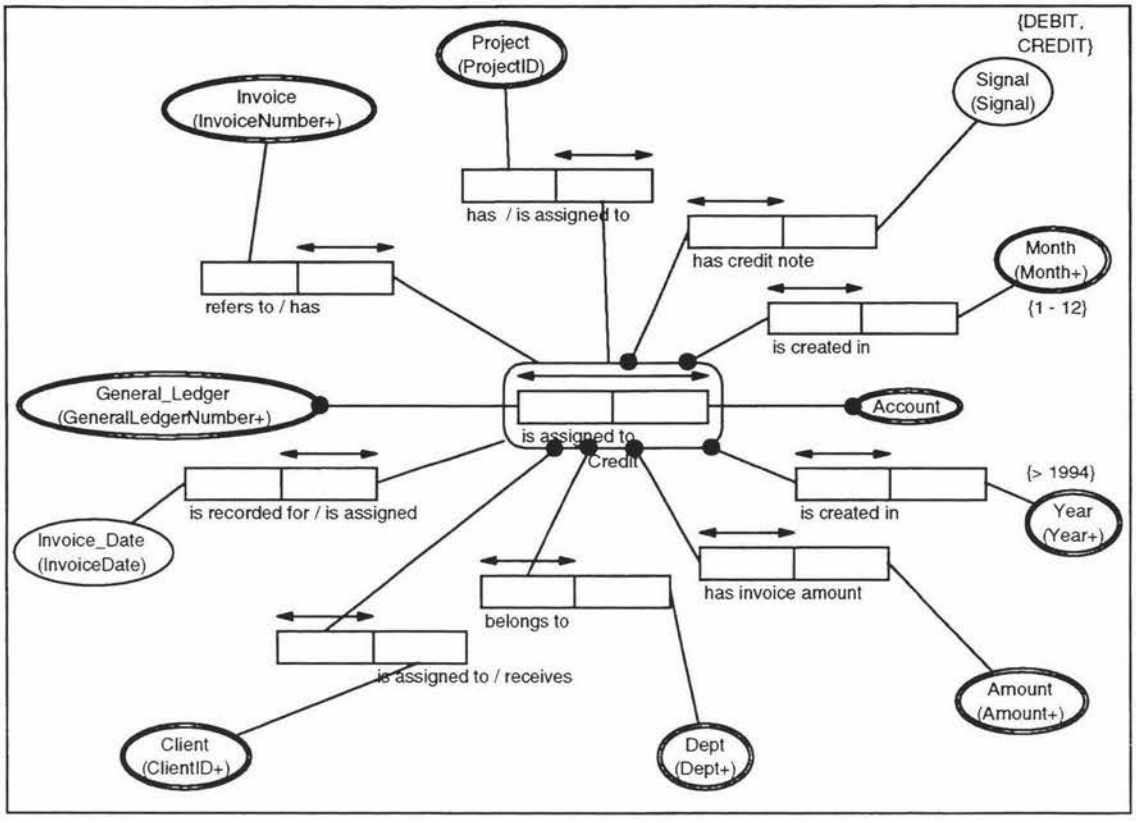


Figure 41. Credits NIAM type diagram with Group B Constructs.

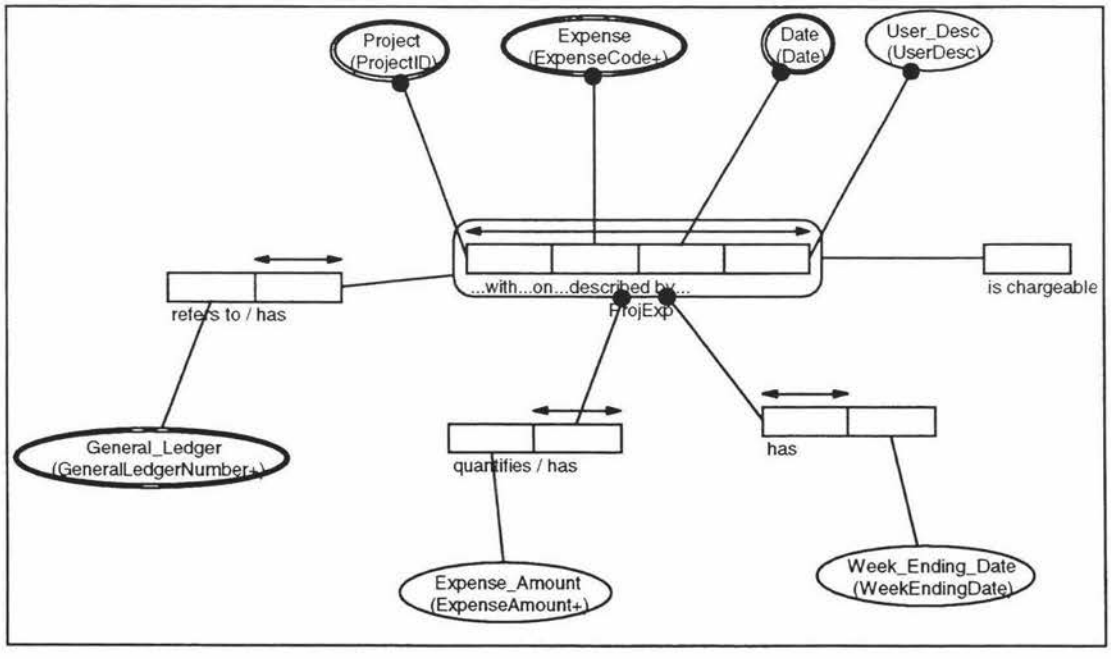


Figure 42. Project Expense NIAM type diagram with Group B Constructs.

Figure 42 does not show depictions of any constructs not identified in the previous two diagrams. It is useful to note that although each of the object types associated with the objectified quaternary fact type *ProjExp* has a mandatory role indicated, it is possible

that in some cases the inclusion of all object types may not be mandatory. This set of constraints is assumed from the database schema and will be confirmed or altered accordingly during the validation stages of the SRDP.

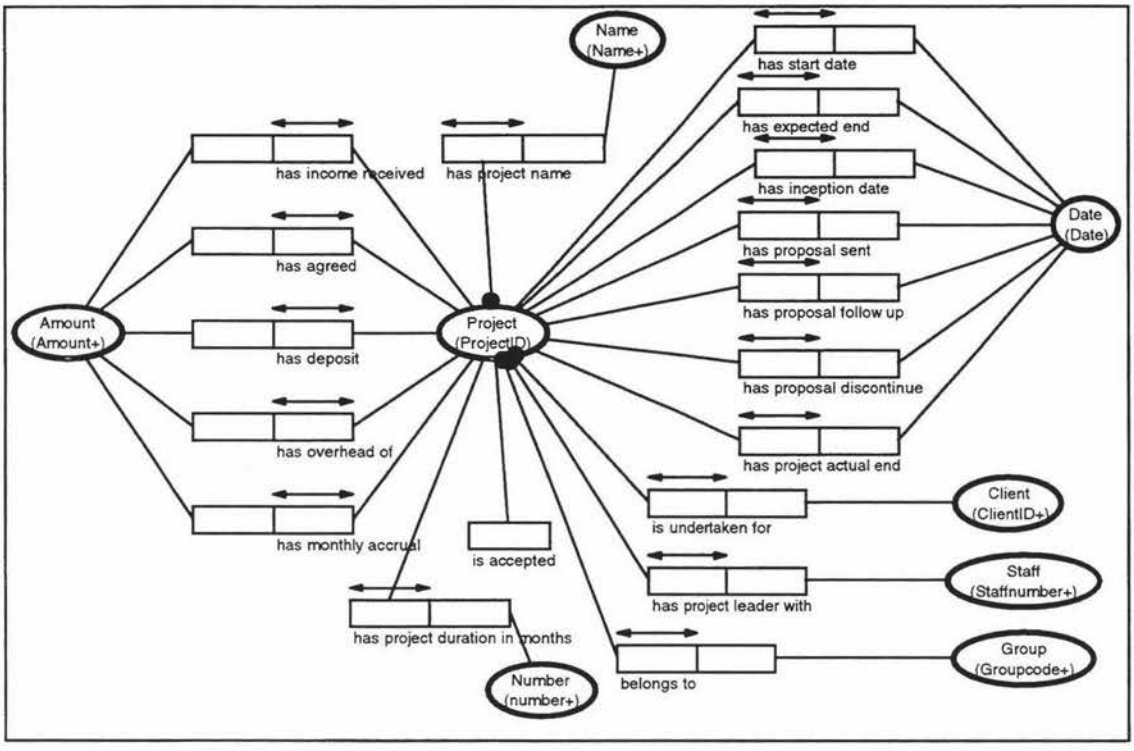


Figure 43. Project NIAM type diagram with Group B Constructs.

Figure 43 is a tidier version of Figure 39 showing the addition of four mandatory role constraints.

8.1.3 Stage 3: Creation of example sentences.

The four diagrams described in the previous two stages contain thirty-eight fact types. These are made up of two unary, thirty-three binary, two objectified binary and one objectified quaternary fact types. For constraint validation purposes it is necessary to draw two values for each object type from the database. The values are then placed in each sentence as per Figure 33 described earlier in this chapter.

An additional issue to be aware of while producing example sentences is the replication of fact types. If these occur they should be noted and taken forward to stage six where they can be reconciled.

A complete set of validation sentences for all the fact types in the four diagrams used in this example requires the production of one hundred and fourteen sentences. Seventeen of these are recorded in the body of this report whilst a complete set can be found in appendix four.

Fact 1. Account number contains *Sub-account / Sub-account* is associated with *Account*

Every *account number* contains at least one *sub-account*
Every *sub-account* is associated with at least one *account number*

Example sentences:
The *account number* with account_number **02020** contains *sub-account* with sub-account number **FORST**.
The *account number* with account_number **02020** contains *sub-account* with sub-account number **NZDB**.
The *account number* with account_number **02030** contains *sub-account* with sub-account number **FORST**.

Fact 2. Account has *Account type.*

Every *Account* has exactly one *Account type*

Example sentences:
The *account* '**02020, FORST**' has *Account type* of account type **Income**.
The *account* '**02020, FORST**' has *Account type* of account type **Expense**.
The *account* '**02030, NZDB**' has *Account type* of account type **Income**.

Figure 44. Fact types from the Account diagram in Figure 39

Fact 3. General ledger number is assigned to Account.

Every *General ledger number is assigned to* at least one *Account*

Example sentences:
The *General ledger number* with number **500** is assigned to *Account* '**02020,FORST**'
The *General ledger number* with number **500** is assigned to *Account* '**02030,NZDB**'
The *General ledger number* with number **600** is assigned to *Account*' **02020,FORST**'

Figure 45. Fact type from the Credits diagram in Figure 40

Fact 4. Project with Expense on Date is described by User Description

Each *Project* with *Expense* on *Date* described by *User Description* is unique.

Example sentences:
The *Project* that has project ID **93-34** with *Expense* that has expense code **46** on *date* **21.11.94** is described by the *User description* that has User_desc **Photocopying**
The *Project* that has project ID **93-34** with *Expense* that has expense code **46** on *date* **21.11.94** is described by the *User description* that has User_desc **Vouchers**
The *Project* that has project ID **93-34** with *Expense* that has expense code **46** on *date* **10.5.95** is described by the *User description* that has User_desc **Photocopying**
The *Project* that has project ID **93-34** with *Expense* that has expense code **52** on *date* **21.11.94** is described by the *User description* that has User_desc **Photocopying**
The *Project* that has project ID **93-66** with *Expense* that has expense code **46** on *date* **21.11.94** is described by the *User description* that has User_desc **Photocopying**

Figure 46. Fact type from the Project Expense diagram in Figure 41.

Fact 5. Project is accepted
Example sentences: The *Project* that has project ID **93-34** is *active* The *Project* that has project ID **93-66** is *active*

Figure 47. Fact type from the Project diagram in Figure 42.

The example sentences above are produced manually using the semantics in the diagrams plus the data in the database and differ a little in appearance to those found in appendix four that have been generated using the CASE tool Infomodeler.

8.1.4 Stage 4: Verification of example sentences by universe of discourse expert.

In this case the researcher is also acting as the universe of discourse expert. The sets of example sentences have been constructed in such a way that most sets will include sentences that are invalid when considered in conjunction with the valid sentences preceding them. The existing constraint information has been deliberately ignored meaning that for non-objectified fact types at least the second example should be invalid. This mass invalidity is caused by all non-objectified binary fact types seemingly possessing a single role uniqueness constraint.

After verifying the complete set of example sentences invalid sentences are noted. For the four diagrams described, the invalid sentence numbers (refer appendix four) are shown in Table 13.

Account Diagram		Credits Diagram		Project Expense Diagram		Project diagram	
Fact #	Example #	Fact #	Example #	Fact #	Example #	Fact #	Example #
1	2	1	2	1	2	1	2
3	2	2	2	2	2	2	2
5	2	3	2	2	3	3	2
5	3	4	2	4	2	4	2
		5	2			5	2
		6	2			6	2
		7	2			7	2
		8	2			8	2
		9	2			9	2
		9	3			10	2
		10	2			11	2
						12	2
						13	2
						14	2
						15	2
						16	2
						18	2

Table 13. Invalid Example Sentences.

Of the example sentences not appearing in Table 13, fact 4 in the Account diagram and fact 3 in the Project expense diagram are objectifications and the reason they do not appear is that by having uniqueness constraints across all of their roles they comply to the set of example sentences. Fact type 7 on the Credits diagram[14] fails the sentence validation test on sentence 2. The problem here is that General Ledger number 23 has been assigned to two different accounts - something that should not happen in the system. This indicates a uniqueness constraint should be placed on the General Ledger role, therefore invalidating the objectification. This is shown in the next stage.

Fact type 5 in the Project expense diagram and fact type 17 in the Project diagram are unary fact types that comply with their respective example sentence sets. Fact type 2 in the Account diagram also complies with its set of example sentences but is a binary fact type. This indicates that it should have a uniqueness constraint over both of its roles, as shown in the next stage, rather than just the Account role.

Fact type 5 in the Account diagram fails to validate sentences two and three. On reflection the universe of discourse expert has decided that there is no need for a Sub-account description as this is effectively accomplished with the Sub-account number. Therefore this fact can be removed.

Fact type 9 on the Credits diagram and fact type 2 on the Project expense diagram both have two invalid sentences, numbers two and three, meaning that rather than a single uniqueness constraint they should each have two uniqueness constraints across both of their roles.

8.1.5 Stage 5 - Incorporation of additional semantic constraints.

This stage is usually completed by the expert in the universe of discourse. In this case this means the author. In many cases this will mean traditional end users but in some it

[14]This fact type is complex because one of its object types (Account) is an objectified binary fact type itself. However, provided Account is valid Credit can be validated in the same manner as other binary fact types.

will fall on the schema designer who in these circumstances may also be considered a novice designer.

The additions performed at this stage to the four diagrams used throughout this example are few. For the first three diagrams there are no additional constraints from group C to be added. The fourth diagram *Project* has three new constraints derived from knowledge of the universe of discourse as shown in Figure 48.

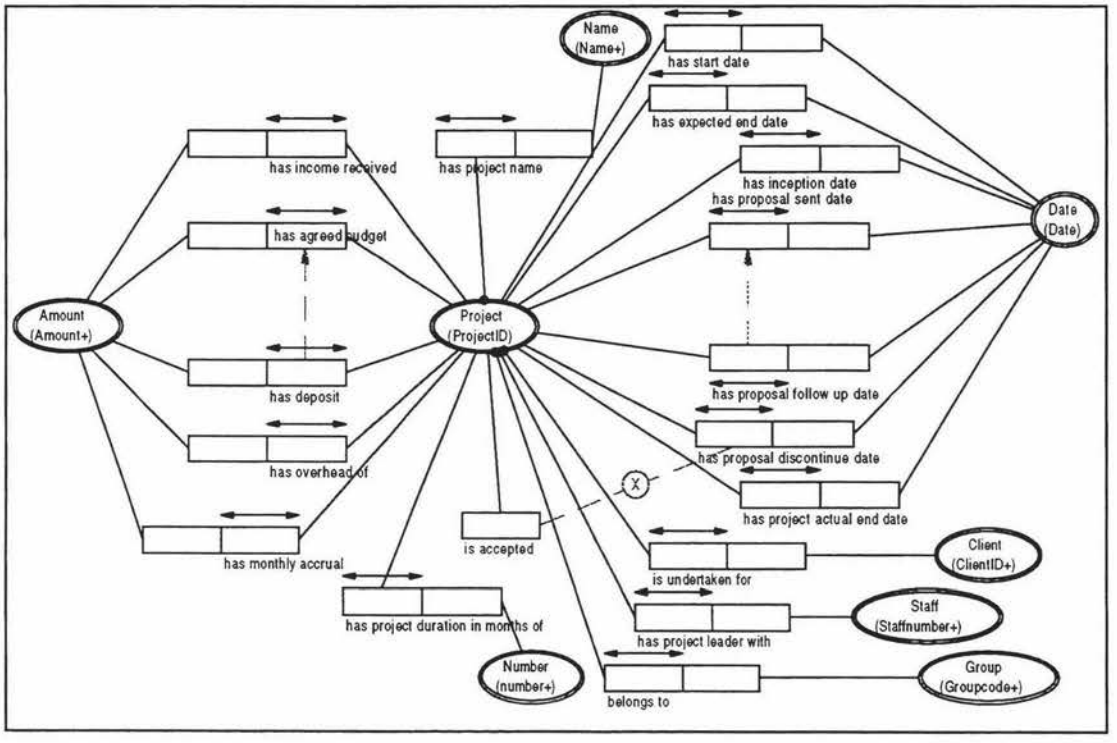


Figure 48. Project Diagram with Group C Constraints.

The two sub-set constraints indicate that the projects that have a deposit amount are a subset of those that have an agreed budget amount and those projects that have follow-up dates are a subset of those that have "proposal sent" dates recorded. The exclusion constraint indicates that if a project is recorded as accepted then it cannot have a proposal discontinue date and vice versa.

8.1.6 Stage 6 - Generation of new logical model and completeness check

The additional constraints identified in stage five and any changes required to validate example sentences that should be valid are included in an updated version of the NIAM type diagrams. These are shown for this example universe of discourse in Figure 49, Figure 50, Figure 51 and Figure 52.

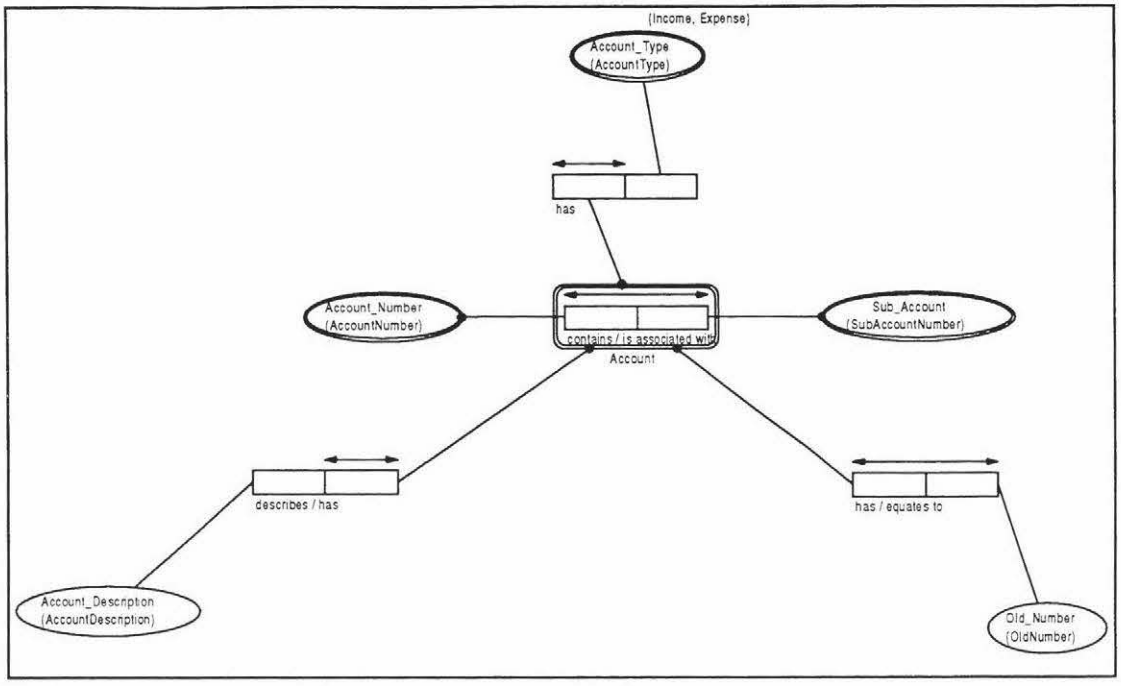


Figure 49. New Logical Model for Account.

Figure 49 shows a revised constraint on the "Account has Old_Number" fact type changing from a single role uniqueness constraint to a constraint over both roles. The fact type involving *Sub-account_description* has been removed for reasons described during the sentence validation process.

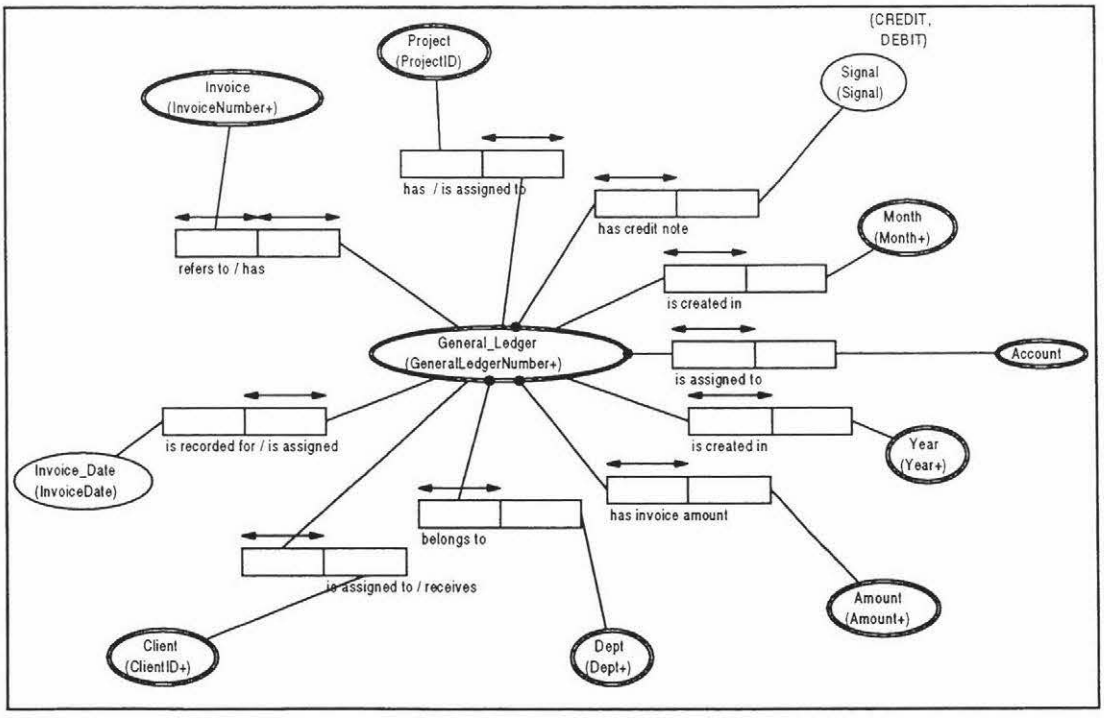


Figure 50. New Logical Model for Credits.

The credits diagram in Figure 50 has undergone a major change as a result of example sentence validation with the dismantling of the objectified fact type and the object type *General_Ledger* now taking a central role in the diagram. This change of emphasis suggests that the diagram should be called the General Ledger diagram rather than Credits. The other change here is the change to the uniqueness constraint of the fact type '*Invoice* refers to *General Ledger*' as indicated by sentence validation. Because of the fundamental change to this diagram it has been necessary to produce a new set of example sentences for validation. These can be found in appendix five.

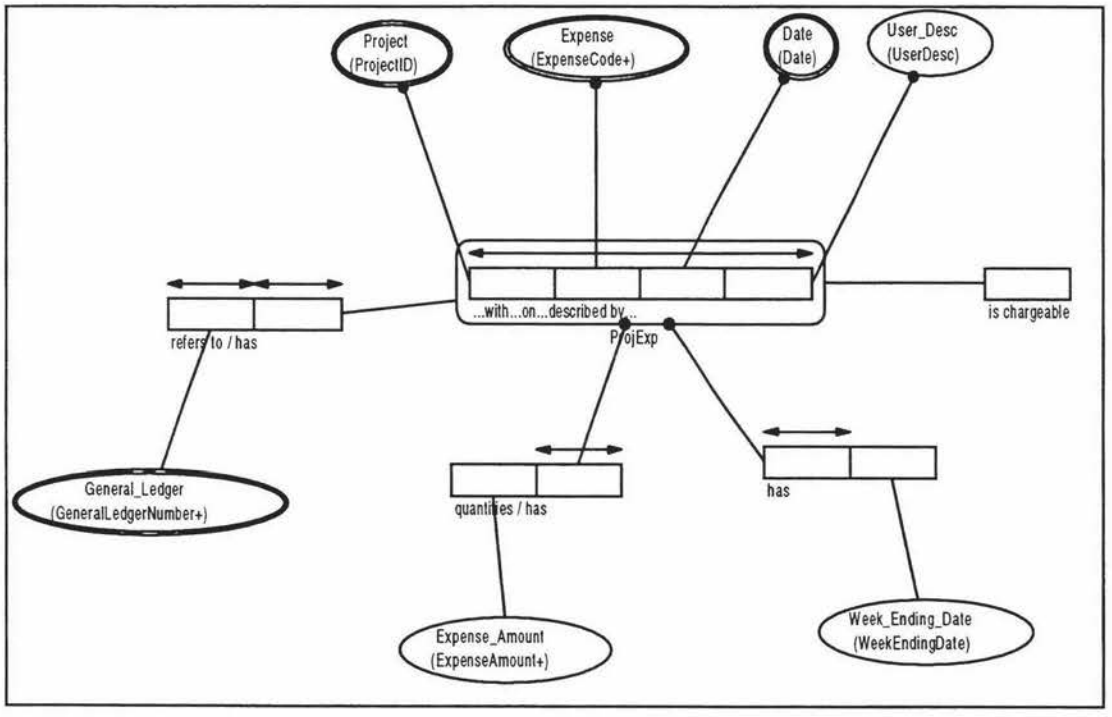


Figure 51. New Logical Model for Project Expense.

The Project Expense diagram in Figure 51 contains only one change arising from sentence validation: a new uniqueness constraint on '*General Ledger* refers to ProjExp'.

The Project diagram in Figure 52 still shows the exclusion and sub-set constraints identified earlier but has not required alterations resulting from sentence validation.

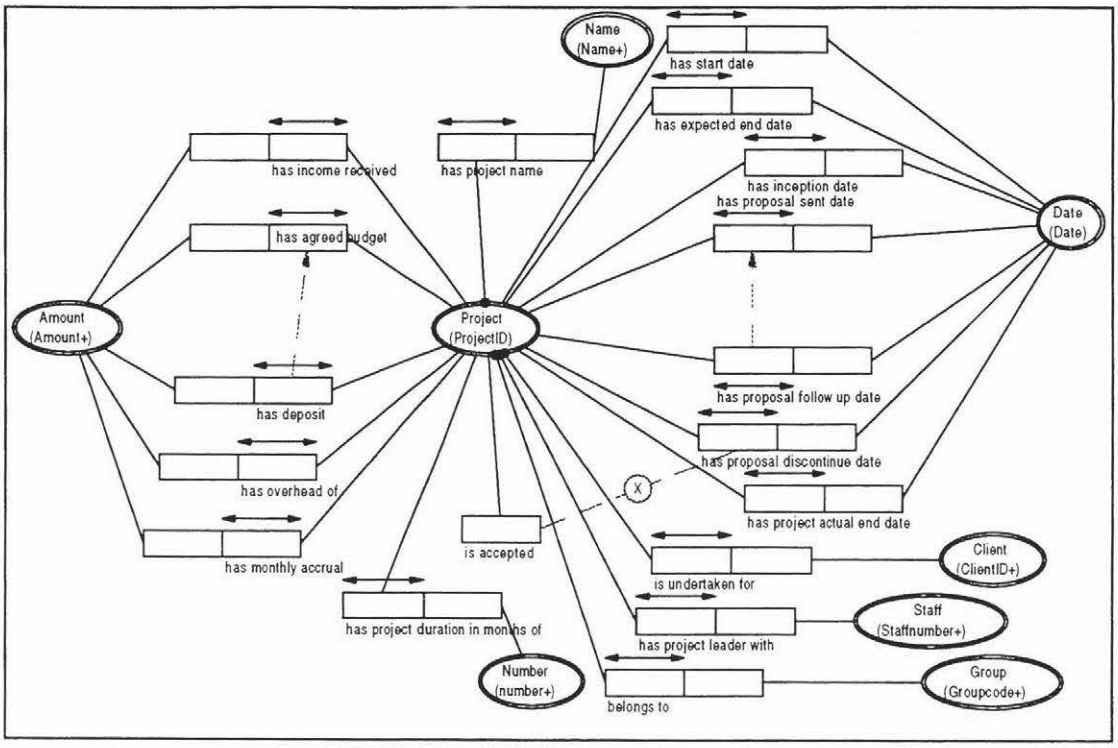


Figure 52. New Logical Model for Project.

Stages three and four of the schema re-design process should be traversed again for the revised fact types with example sentences being constructed and validated by the universe of discourse expert.

Any replicated fact types identified should be reconciled at this stage. This will usually result in the loss of some attributes from the original schema or a reduction in the number of tables to be generated in stage seven.

Because of the limited nature in this example the completeness checks cannot be fully performed at this stage. The complete model of this universe of discourse translates into twenty NIAM type diagrams many of which would add little extra to the exposition of the five stages reported thus far. The two areas where the limited scope of this example appear are; that it becomes impossible to check that the model adequately captures all aspects of the universe of discourse, and it makes for limited identification of replicated object and fact types

8.1.7 Stage 7 - Generation of new physical schema (relational).

Stage seven, the final stage of the schema re-design process, uses the ONF algorithm to map the new logical schema diagram into a revised relation schema. The resultant schema, although often referred to as the implementation schema will be in a highly normalised state and consequently may require some de-normalisation for functional considerations before implementation.

The structure of the revised *Account* table is shown in Table 14. The objectified fact type maps to the primary key fields of the table. It also acts as a foreign key to the *General Ledger (credits)* table that makes it a candidate for participation in a table look-up validation check. Each of the other fact types map to table attributes.

Field Name	*Field Type*
Account number	A5*
Sub account number	A5*
Account Type	A8*
Account description	A30*

Table 14. Revised Account Table Structure.

The fact type "Account has Old number", because of it uniqueness constraint, maps to a new table called *Old Account number* the structure of which is shown in Table 15.

Field Name	*Field Type*
Account number	A5*
Sub account number	A5*
Old number	A45*

Table 15. New Table 'Old Account Number'.

Field Name	*Field Type*	*Foreign Key*
General ledger number	S*	N
Account	A5	Y (Account)
Sub-account	A5	Y (Account)
Invoice	N	Y (Invoice)
Project ID	A8	Y (Project, ProjExp)
Credit note signal	A6	N
Month	S	N
Year	S	N
Invoice amount	S	N
Department	S	N
Client ID	N	Y (Client)
Invoice date	D	N
Expense code	S	Y (Projexp)
ProjExp date	D	Y (ProjExp)
User description	A15	Y (ProjExp)

Table 16. General Ledger Table Structure.

The revised structure of the *Credits* table now renamed *General Ledger* is shown in Table 16. The *General Ledger* attribute now makes up the whole key for this table. There are eight attributes involved in foreign key associations that warrant consideration of table look-up validation checks to manually enforce referential integrity.

For exposition purposes the foreign keys are indicated in a third column in Table 16 along with their respective tables some of which represent areas of the universe of discourse not included in this example.

Field Name	*Field Type*
Project ID	A8*
Expense code	S*
ProjExp date	D*
User description	A15*
Expense amount	$
Week Ending Date	D
Chargeable signal	A1

Table 17. Revised ProjExp Table Structure.

The structure of the *Project Expense* table is displayed in Table 17. This structure is similar to the original except for the exclusion of the *General Ledger* attribute. The new type of constraint on the fact type "General Ledger refers to ProjExp" is implemented by the primary key's constituent attributes acting as foreign key attributes in the *General Ledger* table.

Field Name	*Field Type*
Project id	A8*
Project name	A25
Start date	D
Expect end date	D
Agreed budget amount	$
Deposit amount	$
Client id	N
Income received	$
Project leader	N
Inception date	D
Proposal sent date	D
Proposal follow up date	D
Proposal discontinue date	D
Group code	N
Project actual end date	D
Overhead amount	$
Project acceptance	A1
Monthly accrual amount	$
Project duration	S

Table 18. Structure for Project Table.

The *Project* table in Table 18 has the same structure as the original table. It does however contain three constraints that were unknown at the beginning of the process. Although these constraints cannot be implemented by the DBMS in the database schema the designer should now be aware of them and should be able to implement them through the application code.

8.2 Discussion and comparison with associate researchers' results

Although primarily meant for explanation purposes the study of this database has another aspect to it that, while alluded to previously, requires further discussion. This is the simultaneous modelling of the universe of discourse by an experienced analyst using the NIAM methodology and the Infomodeler CASE tool.

The results of this second modelling experience are reproduced in this thesis in appendix five. They contain NIAM type diagrams and database schema definitions for the complete universe of discourse. Because what has been described in the previous sections represents only part of the universe of discourse it is not possible to do a thorough comparison between the resulting two models, indeed such a comparison will be made on the present report's main experiment in the next chapter. There are however a few issues regarding the variances in outcomes to discuss at this point.

The first point to make is that it is unlikely that the outcome will be identical. Although one of the NIAM methodology's claims is that the CSDP and the ONF algorithm produce predictable and reproducible results (Nijssen & Halpin, 1989) this assumes that the inputs and the starting points are the same. In the case of the present universe of discourse neither of these assumptions hold true. The starting points differ in that the author had an existing database schema from which to derive information while the second researcher began with a set of forms and reports and an interview with the universe of discourse expert. Inputs to model development also differed as the author acted to a large extent in isolation using innate knowledge of the universe of discourse to develop and refine model components at each stage while the second researcher queried three people each with a strong familiarity of the universe of discourse.

Given these differences the results obtained bear strong similarities. The author's model of the complete universe of discourse contains 20 diagrams which map to 18 schema tables containing a total of 145 attributes while the associate researcher's model consists of 11 diagrams which map to 19 schema tables containing a total of 106 attributes. The key structures of twelve tables are equivalent between the two models with four of the remaining nine having close similarities.

A complete set of logical schema diagrams and the table structures they map to are attached in appendix six. Both researchers made use of the Infomodeler CASE tool to facilitate the modelling and design processes. However the current version of Infomodeler does not explicitly support schema re-design making the author's use of the tool less productive than the use made by the associate researcher.

The table structures emanating from the schema re-design process for this universe of discourse are not on the whole, radically different from the original table structures. The changes that have taken place improve the logical design of the database schema and do so in such a way as to make it possible to migrate the existing data from the original schema to the new. This data migration although possible is likely to still contain complications and difficulties, the implications of which are beyond the scope of present report.

SECTION SIX

EXPERIMENT

9 MARXIST experiment

9.1 Experiment methodology

This chapter of the research is loosely termed experimental. It differs from a classical experimental methodology in that seeks to confirm and extend the SRDP rather than disconfirm it (Markus, 1989). The methodology contains elements of the case study research strategy, in that it uses as its primary vehicle one of the case studies identified earlier. The case study method is also incorporated in the research methodology but differs from a classical case study by incorporating a control element (Yin, 1989), which in the present research is an independently administered experiment process. The involvement of the researcher in the process also suggests that the process could be considered as action research. However it can be seen to deviate from action research because the researcher acts in a mechanistic role while at the same time withholding all of their expertise from the process. This is contrary to action research where the contribution of the researcher's expertise is an important component (Yin, 1989).

In broad terms the methodology requires the analysis of a database that is representative of the class of database being researched. The SRDP, in combination with a knowledge of the universe of discourse, produces a revised database design. The revised database design is compared with the original to identify similarities and differences.

The database selected needs to meet three criteria; 1) that it be built and used by a novice designer, 2) that it be of a suitable size and complexity for the research, and 3) that the novice designer is available to act as the universe of discourse expert and provide the feedback required in stages three and four of the SRDP.

Once the candidate database is selected it is necessary to obtain a copy of the complete database schema and at least part of the database's data. The structure is required to

feed into the schema re-design process and the data is required to construct realistic example sentences for validation purposes.

Because the universe of discourse expert is only actively involved in one stage of the schema re-design process the researcher will also need to acquire some knowledge of the universe of discourse. This can be gained from interviews and subsequent discussions with the novice designer. The universe of discourse knowledge should be of sufficient quality to enable the researcher to formulate predicates for the example sentences so that those sentences form lucid English statements in the context of the universe of discourse.

At this point the schema re-design process begins. The first three stages of the SRDP culminating in the production of representative sets of example sentences can be undertaken in succession.

Before seeking validation of the sentences by the universe of discourse expert, an experimental control is initiated and advanced to a point where the novice designer has been interviewed and provided sufficient information for an associate researcher to produce an independent database design. In this methodology this takes the form of a conventional NIAM conceptual schema design procedure based on the same universe of discourse. This control schema is directly compared with the SRDP's schema at the conclusion of the experiment. It is important during the experimentation time that the two researchers do not discuss their findings until each have completed their database design.

Once the universe of discourse expert has been interviewed by the associate researcher the example sentences generated by the first researcher can be verified. Ideally each researcher interacts with different universe of discourse experts to reduce the interference of prior knowledge effecting the expert's input. However because there is only one universe of discourse expert in the present case this interference is unavoidable but is recognised as a possible weakness of the experiment method.

The novice designer may not be able to verify all the example sentences at the first attempt, if this is the case it will be necessary to iterate this phase by generating refined example sentences until all have undergone validation.

Once a completed set of example sentences has been received from the novice designer stages five, six and seven of the schema re-design process are completed. At this point the two database schemata can be compared and assessed for similarities, differences and accurate representations of the universe of discourse.

9.2 Experiment Design

From the four cases studies described, case study D (hereafter referred to as MARXIST) has been chosen for the present experiment. Of the other three cases, one designer was unwilling to participate further, another, while willing and suitable, required the researcher to complete a two hour journey to undertake interviews, consequently it remained the third choice for an experimentation candidate. The third, also willing and suitable remained as a second choice experiment subject should MARXIST have been unable to complete the study.

Because MARXIST is an application developed using Microsoft Access, rather than Paradox as in the previous example, some of the information extracted from the DBMS is different. In particular Access supports a different set of data types (Table 19) to Paradox and rather than implementing user defined referential integrity through the table look-up construct Access does so through cascading updates and deletes.

Data Type	*Definition*
Text	Up to 255 characters
Memo	Up to 64 Kilobytes
Number	Any numeric type
Date/Time	Between years 100 and 9999
Currency	19 digit accuracy
Counter	A number automatically incremented by Access
Yes/No	Logical Yes or No values
OLE Object	A Microsoft Object Linking and Embedding Object

Table 19. Access' Data Type Set (adapted from Access 2's help file).

MARXIST consists of sixteen tables that are described and converted into NIAM type diagrams during stages one and two of the SRDP.

The expected result from this experiment is that the two resultant schemata will contain significant differences. The associate researcher's design is expected to be assessed as a refined and more representative model of the universe of discourse. However both new schemata, when compared with the original design, should possess

relational improvements such as reduced redundancy and be more representative of the universe of discourse.

9.2.1 Stage 1 - Transformation of existing schema to populated diagram with standard predicates.

To avoid over burdening this chapter with diagrams the NIAM type diagrams for this universe of discourse in both stages one and two are shown at the second stage. The schema table descriptions shown in Table 20 through to Table 35 describe the existing groups A constructs (ref Table 4, page 102), each of which can be seen mapped on to Figure 53 through to Figure 68.

Attribute Name	*Type*	*Size*	*Key*
Team ID	Text	3	
Site ID	Text	50	
Repair Agent	Text	50	

Table 20. Defaults Table Structure.

The *Defaults* table, the structure for which is shown in Table 20, is a relatively trivial table containing only one row of data. It has no key structure and consequently when mapped to a NIAM type diagram will be given the broadest constraint possible which in relational terms means placing a uniqueness constraint over all three attributes.

The purpose of the entity is unclear to the author but possibly exists for the purpose of defining default data entry values for the three fields in the application code. All three attributes appear in other entities indicating that this entity may be redundant although its uniqueness constraint indicates that it remains as part of the schema.

Attribute Name	*Type*	*Size*	*Key*
Equip ID	Text	50	P
Equipment	Text	50	F
Usage ID	Number (Integer)	2	

Table 21. Equipment Table Structure.

The *Equipment* entity described in Table 21 seems to be describing some equipment in terms of a code, description and a usage code possibly referencing another entity. Its use may be to identify for what purpose each type of equipment is used. As *Equip ID* acts as a simple primary key in this entity and it appears elsewhere in the schema the data in this entity might be incorporated into another entity at a later stage.

Attribute Name	*Type*	*Size*	*Key*
Location ID	Text	50	P
Equipment	Text	50	

Table 22. Labels Location Table Structure.

The *Labels Location* entity in Table 22 has a very simple structure of just two attributes one of which is a single attribute primary key. Its purpose seems unclear as its current structure will only allow one *equipment* to be recorded for each location. This seems unlikely unless *location* refers to a particular space on a particular shelf in a particular building. The data in this entity may also be incorporated into the *Large Modules* entity at a later stage.

Inspection of the entity's contents reveals that it contains no data which may indicate that the entity is a temporary table used for functional purposes or that the entity is no longer used at all by the system. There is no way of knowing for certain what the entity's purpose is without asking the universe of discourse expert.

It seems likely that the data in this entity will not be mapped into a re-designed schema without some clarification of its role.

Attribute Name	*Type*	*Size*	*Key*
Serial Number	Text	50	P
Type	Text	50	

Table 23. Labels Module Table Structure.

The *Labels Module* entity in Table 23 has a single attribute primary key and one descriptive attribute that appears to also occur in the *Large Modules* entity. The serial number attribute also appears in the *Large Modules* table and it seems likely that the information captured in this entity will also exist in the *Large Modules* table. This entity too contains no data and may be part of an old version of the schema that is no longer used.

Attribute Name	*Type*	*Size*	*Key*
Serial Number	Text	12	P
Type	Text	50	
Location ID	Text	50	
Owner	Text	50	
Status	Text	50	
History	Memo		
Manf Serial	Text	50	
Past Locations	Text	255	
Manf Name	Text	50	
Part Number	Text	50	
Usage	Text	50	
Bar Code	OLE Object		
Model	Text	50	

Table 24. Large Module Labels Table Structure.

The *Large Modules* entity in Table 24 is the third entity in the schema containing no data. It is possible that the previous three entities are all leftover from an iterative development process and therefore are not likely to appear in this form in the revised schema. The entity appears to carry a considerable number of attributes and possibly was intended to act as the main data storage area for the system. Again it has a single attribute primary key and a large number of attributes replicated in other entities.

Attribute Name	*Type*	*Size*	*Key*
Location ID	Text	50	P
Site ID	Text	50	
Equip ID	Text	50	
Usage ID	Number (Integer)	2	
Print Label	Yes/No	1	

Table 25. Locations Table Structure.

The *Locations* entity in Table 25 has a simple key structure but unlike the previous tables it does contain data. It consists of foreign key attributes to three other entities and a logical field to record whether a label has been printed for that particular location.

Attribute Name	*Type*	*Size*	*Key*
Manf ID	Number (Integer)	2	P
Manf Name	Text	50	

Table 26. Manufacturers Table Structure.

The *Manufacturers* entity in Table 26 appears to be a code - decode table. It assumes that rather than entering the manufacturer's name throughout the database the code is entered. This entity is then joined to queries to extract the manufacturer's name for reporting purposes. The *Manf ID* attribute appears in several places in the database

while *Manf name* only appears here, supporting the assumption of a code - decode table function. However its nature may change later with the addition of other attributes such as address and telephone number.

Attribute Name	*Type*	*Size*	*Key*
Module ID	Number (Long)	4	P
Type	Text	50	
Part Number	Text	50	
Model	Text	50	
Repair Agent	Text	50	
Usage ID	Number (Integer)	2	
Manf ID	Number (Integer)	2	
Type No	Number (Integer)	2	
Use Manf	Number (Double)	8	

Table 27. Module Descriptions Table Structure.

The *Module Descriptions* entity in Table 27 seems a little more substantial than most of the previous ones mentioned. This entity also contains significant quantities of data that indicates it may be an important table in the schema. The entity seems to describe an electrical component in terms of its identifying codes and numbers as well as where it has come from and who is responsible for its repair.

Attribute Name	*Type*	*Size*	*Key*
Serial Number	Text	12	P
Module ID	Number (Long)	4	
Location ID	Text	50	
Past Locations	Text	255	
Owner	Text	50	
Status	Text	50	
Print Label	Yes/No	1	
History	Memo		
Serial	Number (Integer)	2	
Manf Serial	Text	50	

Table 28. Modules Table Structure.

Table 28 appears to show information about instantiations of the type of equipment recorded in the previous entity. The *Serial number* identifies an individual occurrence of a component to the database. This attribute appears to be system generated.

Attribute Name	*Type*	*Size*	*Key*
Owner	Text	50	P
Address 1	Text	50	
Address 2	Text	50	
Town	Text	50	
Phone	Text	50	
Fax	Text	50	

Table 29. Owners Table Structure.

The entity shown in Table 29 records data about people or organisations involved in the system. It is identified by a single attribute key *Owner* that appears to be an abbreviation of a company name. The two address lines appear to be free format allowing the users to put whatever data they wish in them.

Attribute Name	*Type*	*Size*	*Key*
ID	Text	3	
Name	Text	20	
Password	Text	20	
Team ID	Text	3	

Table 30. Person Table Structure.

The structure of the *Person* entity shown in Table 30 appears to be exist to restrict access to the database application. It appears to identify people by their *ID*, although this is not denoted as the primary key, but it has no relationship with any other entity in the database except for a foreign key relationship with the *Team* entity through the *Team ID* attribute.

Attribute Name	*Type*	*Size*	*Key*
Repair Agent	Text	50	P
Address 1	Text	50	
Address 2	Text	50	
Town	Text	50	
Phone	Text	50	
Fax	Text	50	

Table 31. Repair Agents Table Structure.

As with the *Owners* entity in Table 29, Table 31 appears to record information about people or organisations. Its descriptive attributes are the same as found in the *Owners* entity except for the primary key attribute which in this case is called *Repair agent*. It seems likely that the attributes *Repair agent* and *Owner* are both taken from the same domain. If this is the case it is also possible that these two entities should be merged into one.

Attribute Name	*Type*	*Size*	*Key*
Site ID	Text	50	P
Site Name	Text	50	
Team ID	Text	50	

Table 32. Sites Table Structure.

The *Site* entity in Table 32 appears to be a small entity consisting only of a primary key, a descriptive attribute and a foreign key to the *Teams* entity. The presence of an entity with this structure enables *Teams* to be assigned to multiple *Sites*. However it does restrict a *site* to only one *team*.

Attribute Name	*Type*	*Size*	*Key*
Team ID	Text	50	P
Teams Name	Text	50	
Address 1	Text	50	
Address 2	Text	50	
Town	Text	50	
Phone	Text	50	
Fax	Text	50	
Manager	Text	50	

Table 33. Teams Table Structure.

The structure of the *Teams* entity shown in Table 33 has similarities with the *Owner* and *Repair agent* entities discussed earlier. It does however have different attributes, namely *Manager* and *Team name* as well as a different primary key, *Team ID*.

Attribute Name	*Type*	*Size*	*Key*
Serial Number	Text	50	P
Location ID	Text	255	
Status	Text	255	
Date	Date/Time	8	
Comment	Text	255	

Table 34. Update Table Structure.

From the name of the entity in Table 34, *Update*, it seems reasonable to assume that this exists for a functional reason. Having the same primary key as a previous entity indicates that if this is not the case the two entities might be joined.

Attribute Name	*Type*	*Size*	*Key*
Usage ID	Number (Integer)	2	P
Usage	Text	50	

Table 35. Usage Table Structure.

The usage entity in Table 35 seems to be a code - decode table attached to the *Module descriptions* entity.

9.2.2 Stage 2 - Addition of meta-data from the database to explicitly show constraints.

The diagrams shown in stage two represent both the A and B (refer Table 5, page 104) constructs extracted from the original database schema. The differences between what the diagrams look like in stage one and stage two are minor for this universe of discourse and therefore for efficiency considerations stage one diagrams have been omitted in the present report.

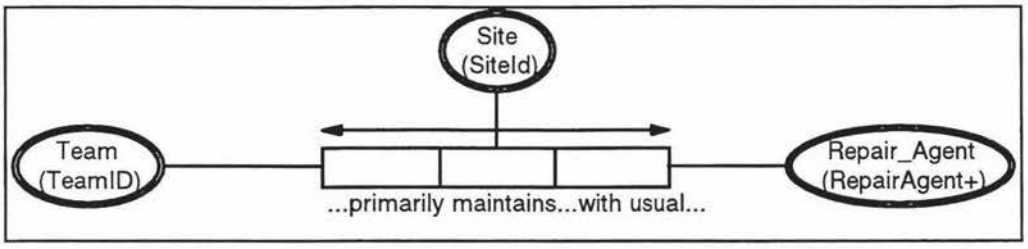


Figure 53. Schema Diagram of Defaults Table

The diagram in Figure 53 depicts the *Defaults* entity which has no assigned primary key. The uniqueness constraint extends across all three attributes meaning that a combination of these three attributes must be unique. There are no group B constructs explicitly included in the DBMS schema and it does not seem reasonable to assume that any object type has a mandatory role in the fact type. Each of the object types is replicated in other diagrams; the *Team* object in the *Person*, *Sites* and *Team* diagrams; the *Site* object in the *Locations*, and *Sites* diagrams; and the *Repair agent* object in the *Module descriptions* and *Repair agents* diagrams.

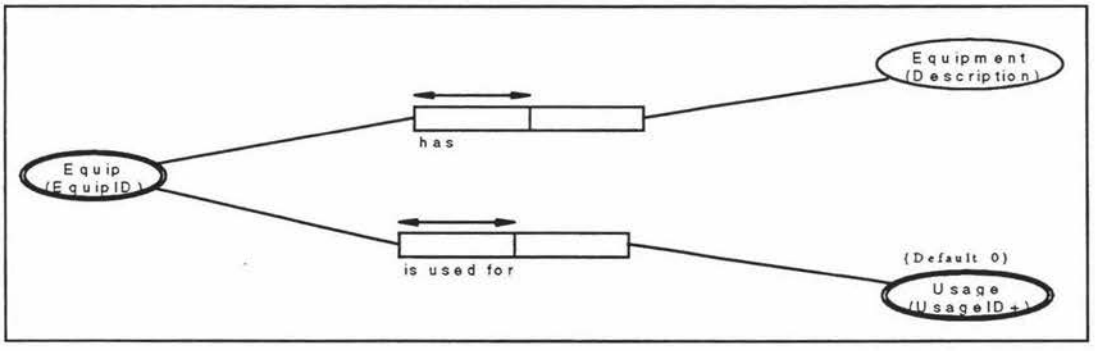


Figure 54. Schema Diagram of Equipment Table.

The database entity represented in Figure 54 is typical of most entities in this universe of discourse in that it contains a single attribute primary key. This key is rendered in the diagram by its participation in binary fact types with each of the other objects in the diagram and the placement of the uniqueness constraint on the roles adjacent to the object representing the primary key attribute. The diagram contains three group B constructs drawn from the DBMS, the first being that the object type *Usage* has a default value of '0', and the second and third that there is an many-to-one relationship between the *Equip* and *Usage* tables, and a one-to-many relationship between *Equipment* and *Equip* both indicated by foreign key relationships.

One object type, *'Description'*, is unique to this diagram while the others are found on other diagrams. The *'Equip'* object type is also found on the *'Locations'* diagram and the *'Usage'* object type is replicated on *'Large Module Labels'*, *'Module Descriptions'*, *'Locations'* and the *'Usage'* diagrams.

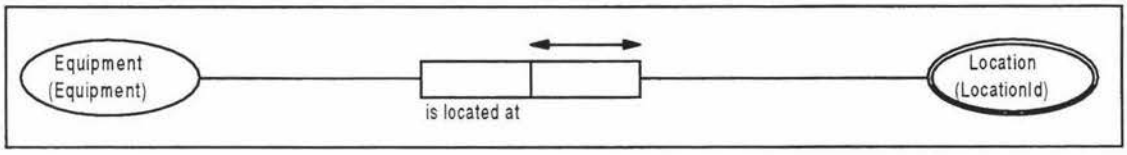


Figure 55. Schema Diagram of Labels Location Table.

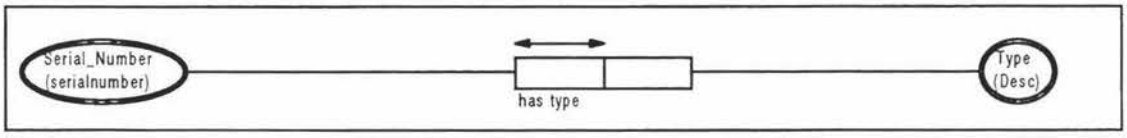


Figure 56. Schema Diagram of Labels Module Table.

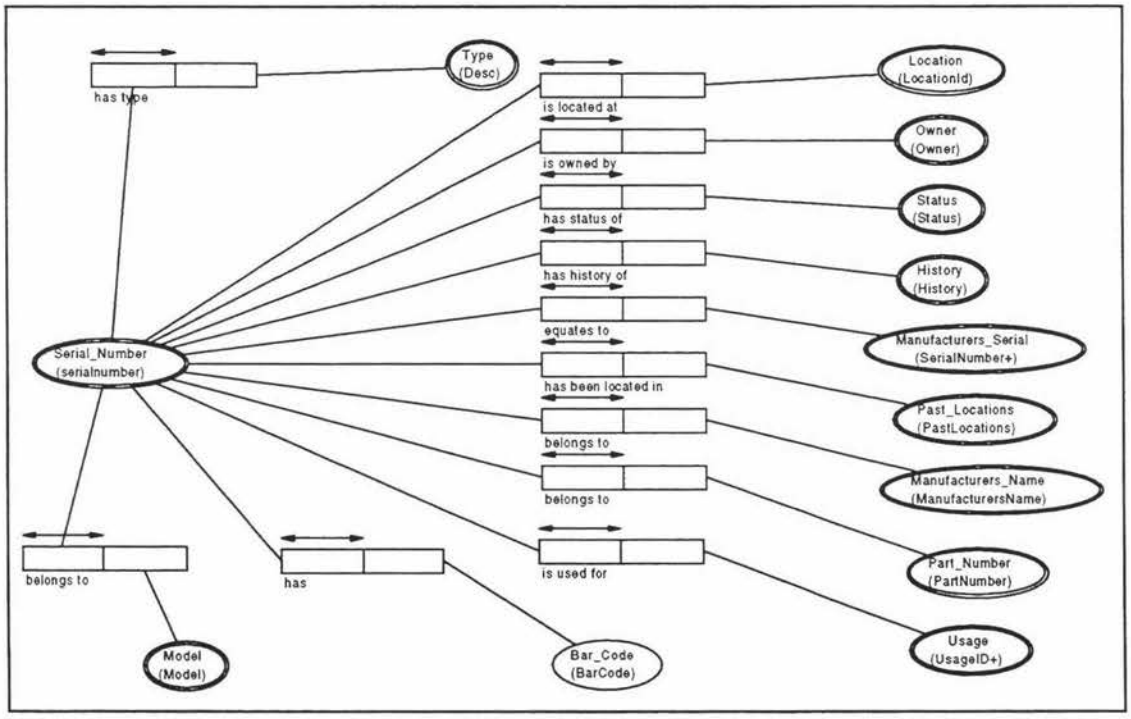


Figure 57. Schema Diagram of Large Module Labels Table.

The three diagrams in Figure 55, Figure 56 and Figure 57 are the three tables in the schema that contain no data. This fact makes them look as though they are no longer used in the design. There are however two object types that do not occur on any other diagram, *'Equipment'* and *'Bar Code'*. The *'Equipment'* object is associated with the *'Location'* object and looks very similar to the fact type involving *'Location'* and *'Equip'* in Figure 58. It seems likely that *'Equipment'* is a synonym for *'Equip'* but as

there is no data in the *'Labels Location'* table it is impossible to confirm this from the information contained in the database schema. There appears to be no direct synonym for the *'Bar Code'* object but it is possible that the *'Print Label'* unary fact type in Figure 61 performs the same function.

The only group B construct present in these diagrams is a one-to-one relationship between the *'Locations'* and *'Labels Location'* tables. Because these tables have the same primary key they could be reduced to a single table if *'Equip'* and *'Equipment'* prove not to be synonymous.

The "Serial number has type Type" fact type is present in Figure 56 and Figure 57 indicating that the occurrence in Figure 56 is redundant. The fact types in Figure 57 with the possible exception of *'Bar Code'* are all depicted either directly or indirectly in Figure 59, Figure 60 or Figure 61. This further supports the previous suggestion that these three diagrams and their respective database tables can be removed from the database schema without compromising the integrity of the data.

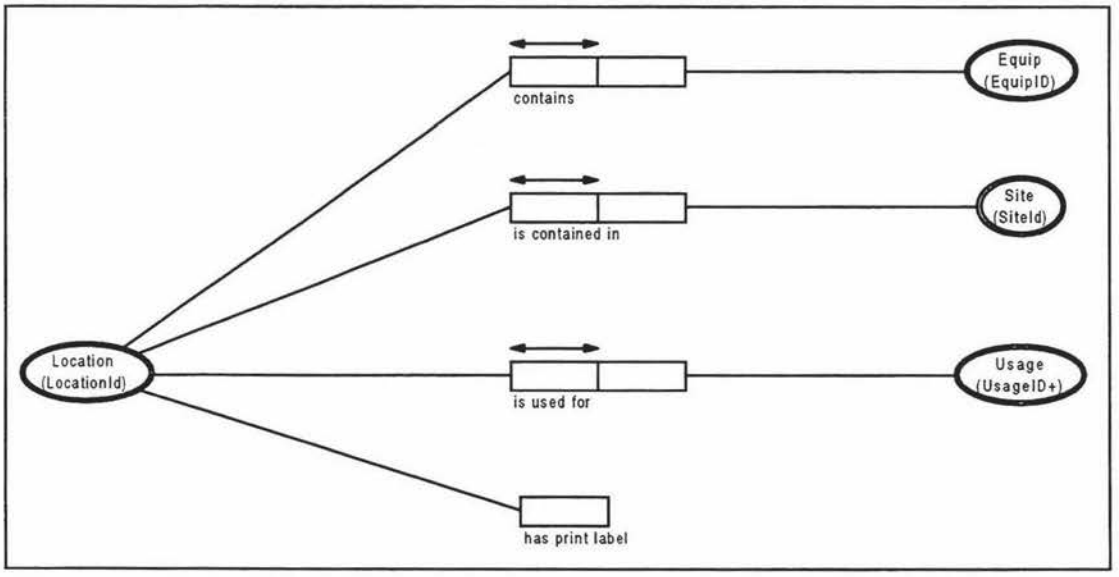


Figure 58. Schema Diagram of Locations Table

As mentioned previously the *'Locations'* table depicted in Figure 58 has a one-to-one relationship with the *'Labels Location'* table. It has another three group B constructs in the form of explicit relationships; a many-to-one relationship to the *'Usage'* table, a many-to-one relationship to the *'Equipment'* table and a many-to-one relationship to the *'Sites'* table. It has a single attribute primary key and all of its object types are replicated on other diagrams. The *'Location'* object type also appears on the *'Labels*

Location', *'Large Module Labels'*, *'Modules'*, and *'Update'* diagrams. The *'Equip'* object type also appears on the *'Equipment'* diagram. The *'Site'* object also appears on the *'Defaults'* and *'Sites'* diagrams. The *'Usage'* object type also appears on the *'Equipment'*, *'Large Module Labels'*, *'Module Descriptions'* and *'Usage'* diagrams.

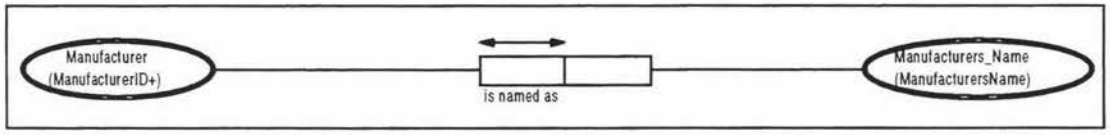


Figure 59. Schema Diagram of Manufacturers Table.

The database table depicted in Figure 59 is a code - decode table. Normally the decode attribute *'Manufacturers Name'* would not appear elsewhere in the schema but in this case it also appears in the *'Large Module Labels'* diagram which has been identified previously as being likely to depict a redundant table. The *'Manufacturer'* object represents the primary key attribute and also appears in the *'Module Descriptions'* diagram. The single group B construct is a one-to-many relationship between the *'Manufacturers'* table and the *'Module Descriptions'* table.

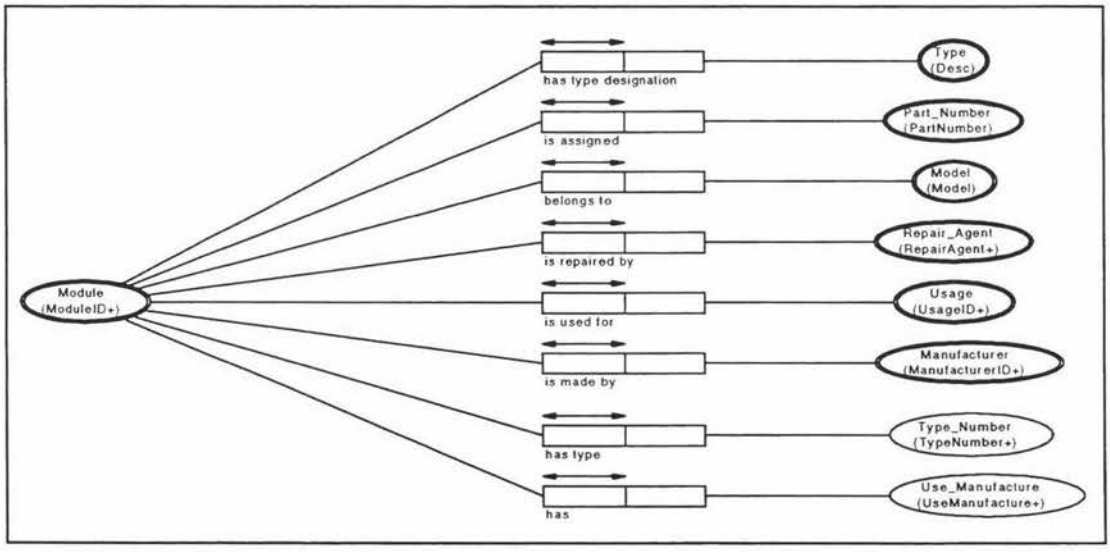


Figure 60. Schema Diagram of Module Descriptions Table.

The *'Module Descriptions'* table depicted in Figure 60 describes a particular equipment type. The diagram does contain two unique object types: *'Type Number'* and *'Use Manufacturer'* with the remainder of its objects appearing on other diagrams.

An examination of the data in this table reveals that the *'Module'* attribute is a concatenation of the *'Use Manufacture'* and *'Type Number'* attributes, with *the 'Use*

Manufacture' attribute itself being a concatenation of the *'Usage'* and *'Manufacturer'* attributes. This being the case the *'Module'* object type could be replaced as the primary key for this table by a unique combination of the *'Usage'*, *'Manufacturer'* and *'Type Number'* object types. The *'Module'* and *'Use Manufacture'* objects would then be completely redundant and could be eliminated from the diagram.

Of the other object types depicted here *'Type'* also appears on the *'Labels Module'* and *'Large Module Labels'* diagrams. The object *'Part Number'* also appears on the *'Large Module Labels'* diagram. The object *'Model'* also appears on the *'Large Module Labels'* diagram. The object *'Repair Agent'* also appears on the *'Defaults'* and *'Repair Agents'* diagram. The object *'Usage'* also appears on the *'Equipment'*, *'Large Module Labels'* and *'Usage'* diagrams.

There are four explicit relationships in the DBMS that are classed as group B constructs indicated by foreign key relationships. These are a many-to-one relationship to the *'Repair Agents'* table, a many-to-one relationship to the *'Manufacturers'* table, a one-to-many relationship to the 'Modules' table and a many-to-one relationship to the 'Usage' table.

The 'Modules' table depicted in Figure 61 records information pertaining to an individual piece of equipment. Its design is similar to the table depicted by the previous diagram in that its key object type *'Serial Number'* is a concatenation of the *'Module'* and *'Serial'* attributes. A similar procedure could be performed on this structure by removing the *'Serial Number'* object and placing a uniqueness constraint over the concatenation of the *'Module'* and *'Serial'* object types.

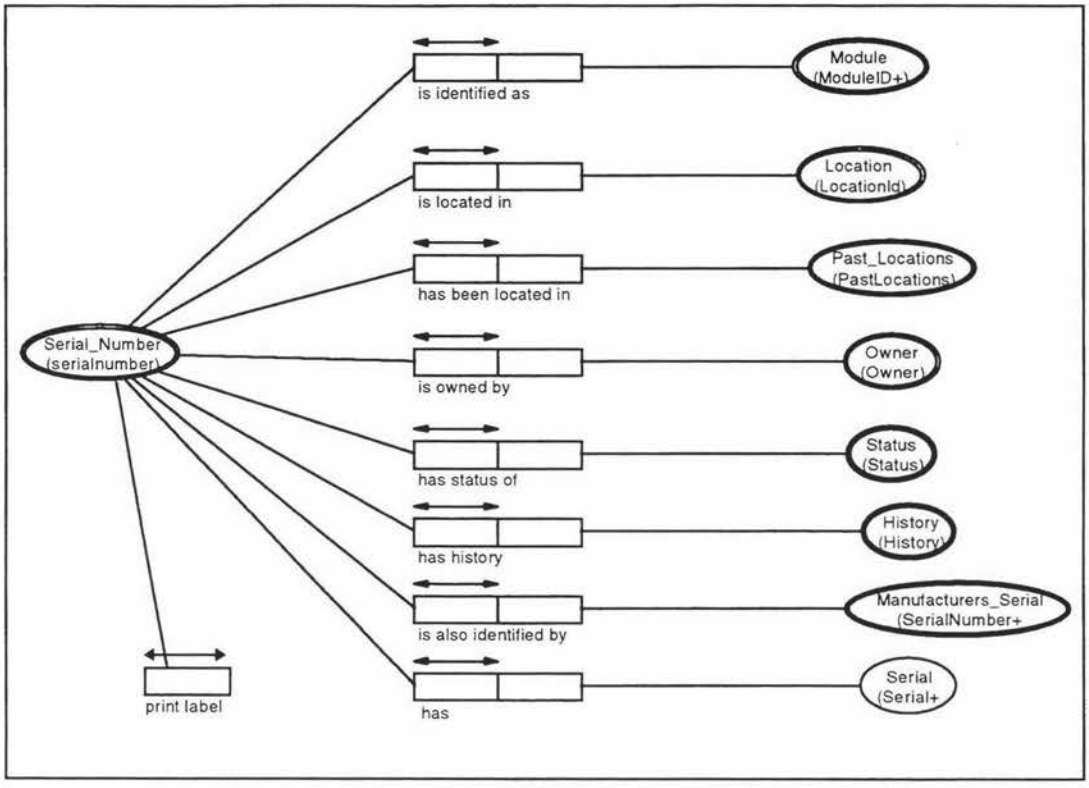


Figure 61. Schema Diagram of Modules Table.

Except for the *'Serial'* object type all objects appear in other diagrams. The object *'Serial Number'* also appears on the *'Labels Module'*, *'Large Modules Labels'* and *'Update'* diagrams. The object *'Module'* also appears on the *'Module Descriptions'* diagram. The object *'Location'* also appears on the *'Labels Location'*, *'Large Modules Labels'*, *'Locations'* and *'Update'* diagrams. The object *'Past Locations'* also appears on the *'Large Modules Labels'* diagram. The object *'Owner'* also appears on the *'Large Modules Labels'*, and *'Owner'* diagrams. The object *'Status'* also appears on the *'Large Modules Labels'* and *'Update'* diagrams. The objects *'History'* and *'Manufacturers Serial'* also appear on the *'Large Modules Labels'* diagram.

There are three foreign key relationships in the DBMS for the *Modules* table. A many-to-one relationship with *'Owners'*, a many-to-one relationship with *'Module Descriptions'* and a many-to-one relationship with *'Locations'*.

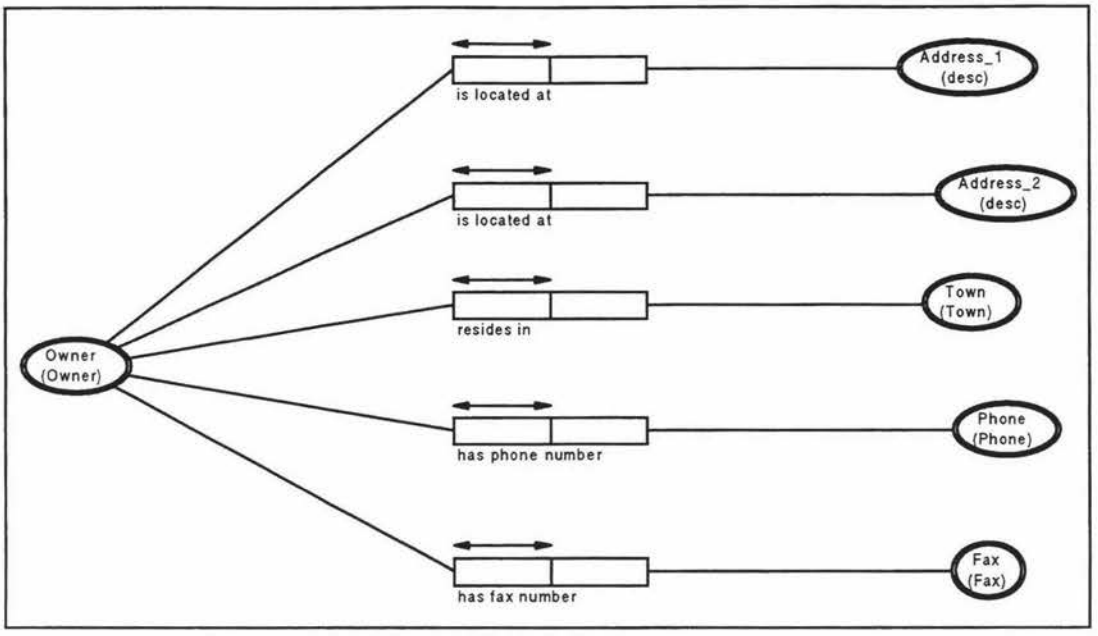


Figure 62. Schema Diagram of Owners Table.

The *'Owners'* table shown in Figure 62 and the *'Repair Agents'* table shown in Figure 63 share a common set of attributes with the exception of the primary key. An examination of the data, however, reveals that there is a strong possibility that the *'Owner'* and *'Repair Agent'* attributes are drawn from the same domain. If this is the case then these two tables are effectively the same and can be joined together.

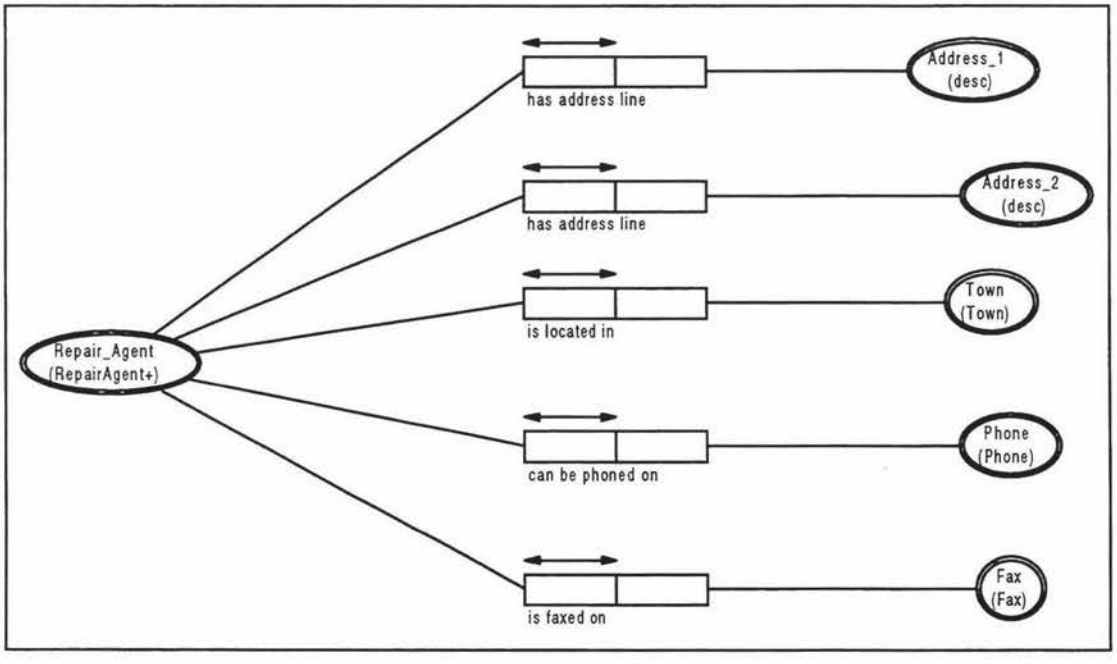


Figure 63. Schema Diagram of Repair Agents Table.

The *'Owner'* table has one foreign key relationship in the DBMS; a one-to-many relationship to *'Modules'*. The *'Repair Agent'* table also has a one-to-many

relationship to the *'Module Descriptions'* table. These relationships might change if the key structures on the *'Modules'* and *'Module Descriptions'* table are altered as discussed previously.

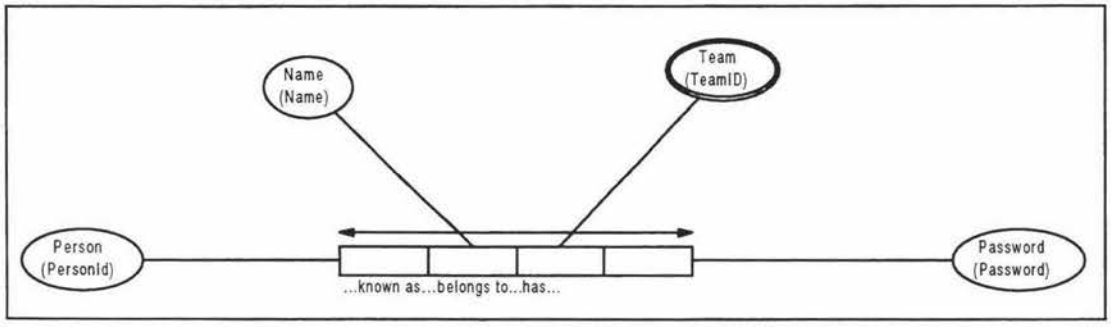


Figure 64. Schema Diagram of Person Table.

Figure 64 shows the diagram for the *'Person'* table in the database schema. There is no primary key defined in the DBMS therefore the uniqueness constraint extends across all four roles in the fact type. There is no group B construct recorded in the DBMS however there is one object type, *'Team'* that appears elsewhere. It appears in the *'Defaults'* and *'Sites'* diagrams.

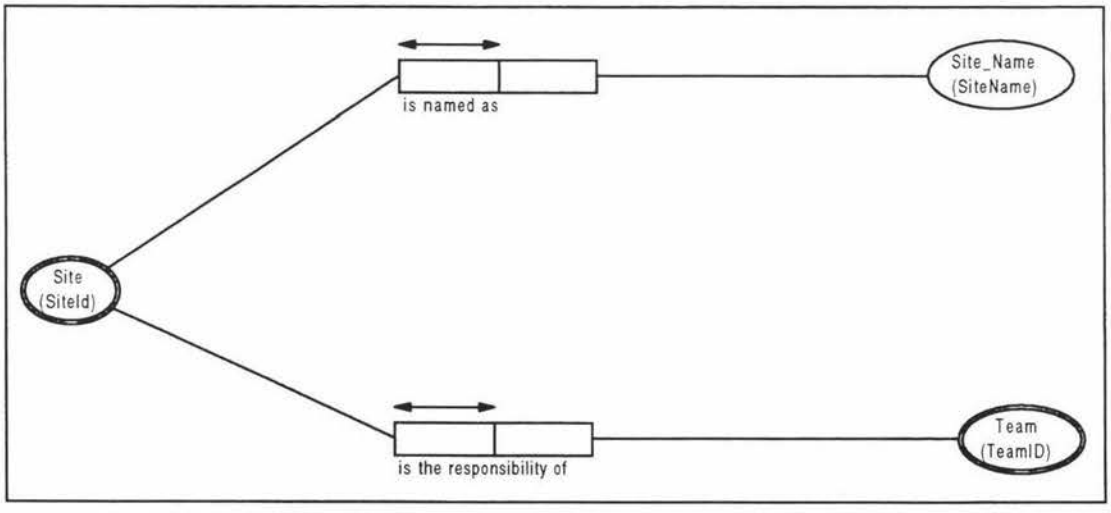


Figure 65. Schema Diagram of Sites Table

The *'Sites'* table depicted in Figure 65 is a code - decode table for the site name and also records which team works at which site. The structure shown restricts each site to the responsibility of only one team. The object *'Site'* also appears on the *'Locations'* and *'Defaults'* diagrams. The object *'Team'* also appears on the 'Default', *'Person'* and *'Teams'* diagrams.

There are two group B constructs in the DBMS for the *Sites* table. There is a many-to-one relationship to *'Teams'* and a one-to-many relationship to *'Locations'*.

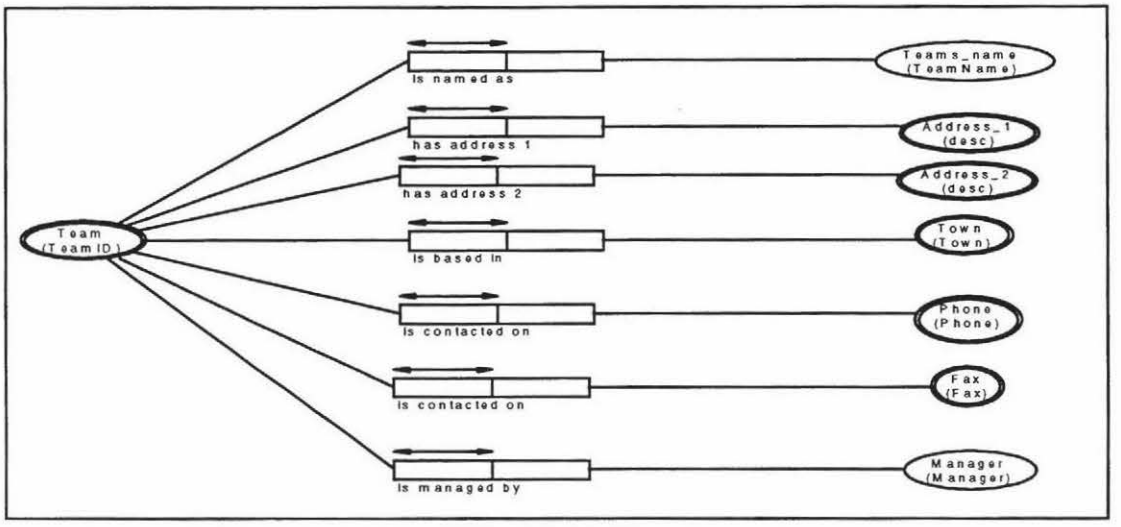


Figure 66. Schema Diagram of Teams Table.

The *'Teams'* table depicted in Figure 66 records information about the different teams involved in this universe of discourse. The structure has a lot in common with the *'Repair Agent'* and *'Owner'* diagrams possessing replications of five objects found in these diagrams. It also has two objects only found on this diagram; *'Teams Name'* and *'Manager'*. The *'Team'* attribute is a three letter acronym representing the full name of that team. This diagram has one group B construct which is a defined one-to-many relationship to the *'Sites'* table.

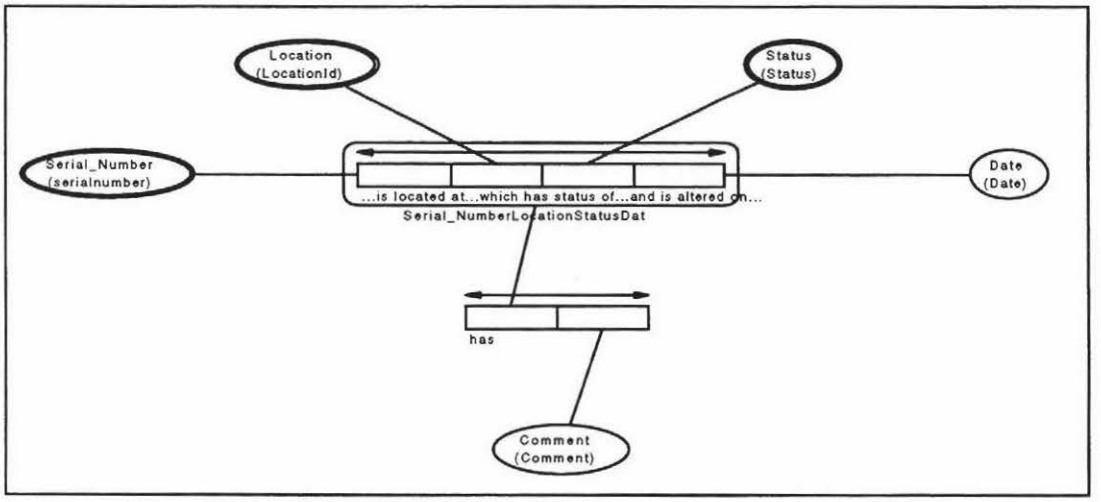


Figure 67. Schema Diagram of Update Table.

At this point the purpose of the *'Update'* table depicted in Figure 67 remains uncertain. It has no defined primary key and no relationships defined in the DBMS.

The diagram shows a key structure comprising of all five attributes that should be shown using an objectified quinternary fact type. In this report the tool used to draw this diagram is unable to produce fact types of arity higher than four therefore the representation is of a binary fact type joined to an objectified quaternary fact type. The combination of the two uniqueness constraints indicates that a combination of all five objects must be unique in the population.

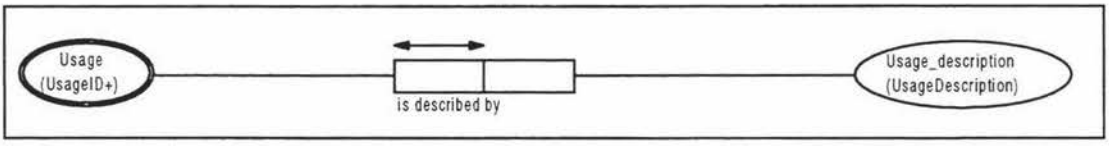


Figure 68. Schema Diagram of Usage Table.

Figure 68 shows another code - decode table, this time for equipment usage. The 'Usage' object is a numerical code that corresponds to a description of the purpose for which it is used. The *'Usage'* object also appears on the *'Large Module Labels'*, *'Locations'* and *'Module Descriptions'* diagrams. The *'Usage Description'* is unique to this diagram. There are also three relationships defined in the DBMS that are classified as group B constructs. These are a one-to-many relationship to the *'Equipment'* table, a one-to-many relationship to the *'Locations'* table and a one-to-many relationship to the *'Module Descriptions'* table.

9.2.3 Stage 3 - Creation of example sentences.

Once Group A and B constructs have been included in the schema diagrams these are used to generate sets of example sentences to be verified by the universe of discourse expert. In this example these have been generated by the CASE tool Infomodeler and amount to sixty-two sets of sentences.

In addition to the sets of sentences, the universe of discourse expert also received a set of definitions for the thirty-nine objects in the diagrams. The full report of object definitions and sentence sets runs for nine pages and has been placed in appendix seven. Described in this part of the report is the feedback received from the universe of discourse expert who in this case is also the original database designer.

9.2.4 Stage 4 - Verification of example sentences by universe of discourse expert.

The verified sets of sentences contained in appendix seven contain the input from the universe of discourse expert. All except three sentence sets are identified as including false sentences by using bold font type (appendix 7). Of those that are assessed as being correct two are unary fact types which by their nature are unlikely to be incorrect and the other is a quinternary fact type that, according to the universe of discourse expert exists in the schema for a specific functional purpose. Those that that have been identified as false are in most instances the second sentence in the set. This indicates that 57 of the binary fact types have a simple functional dependency. Because of the way these fact types interact this means the most of the tables will also have single attribute keys.

Sentence set 22 is a quaternary fact type with three of its sentences noted as invalid. This translates to the existence of a correct uniqueness constraint existing on the object type *'Person'*.

9.2.5 Stage 5 - Incorporation of additional semantic constraints.

Additional semantic constraints such as occurrence frequency constraints and equality constraints have been taken directly from the existing schema. The documentation received from the universe of discourse expert contains solely information concerning the verification of sentence sets and therefore the inclusion of appropriate uniqueness constraints on fact types.

The lack of additional semantic constraints means that, except for semantically derived mandatory role constraints, for this particular case study there is no obtainable output that should be included in the revised logical model.

9.2.6 Stage 6 - Generation of new logical model (NIAM type diagram)

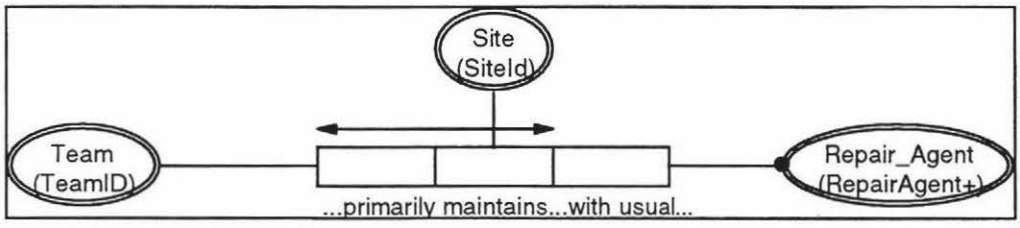


Figure 69. Defaults Table.

In Figure 69 the uniqueness constraint differs from that shown on the corresponding diagram in stage 2 (Figure 53). The constraint now means that each *Team* and *Site* combination must be unique in the table and can only have one *Repair Agent.* A Mandatory participation role is also added here to the *Repair Agent* role meaning that each instance of *Repair Agent* must participate in at least one fact type.

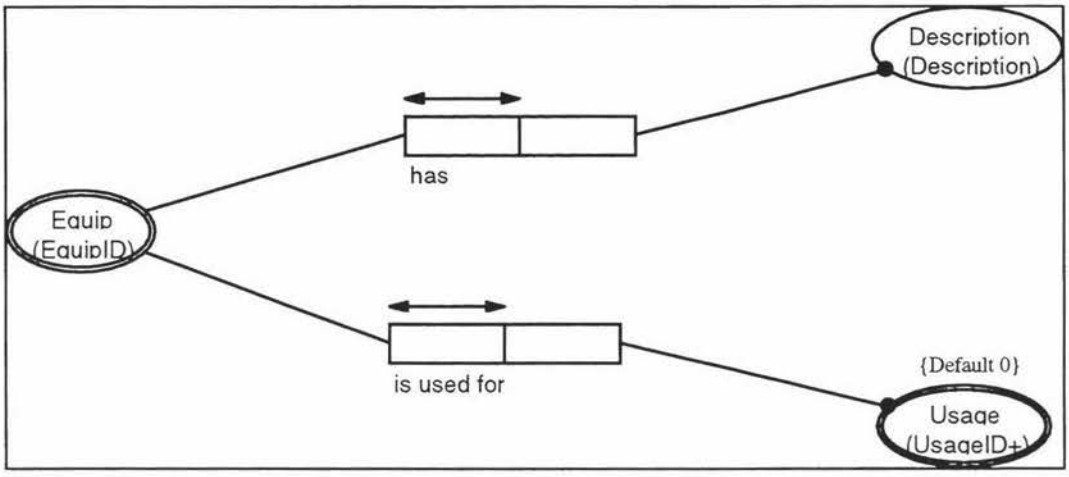


Figure 70. Equipment Table.

In Figure 70 the only addition is the inclusion of mandatory role constraints on the fact types associated with the *Equip* object. The default value of 0 is carried forward from the previous version of the diagram.

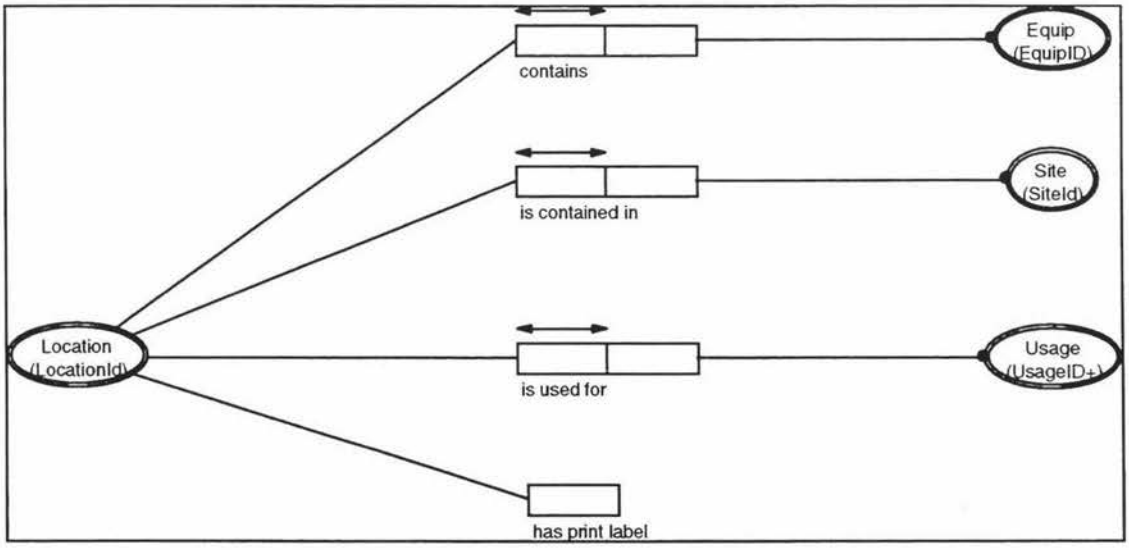


Figure 71. Locations Table

The only additions to the *Locations* table in Figure 71 are the inclusion of mandatory role constraints on the fact types shown.

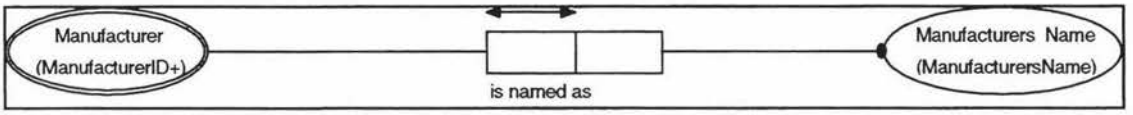


Figure 72. Manufacturers Table.

In Figure 72 the only addition is specification of the mandatory nature of the *Manufacturers Name* object type in the fact type.

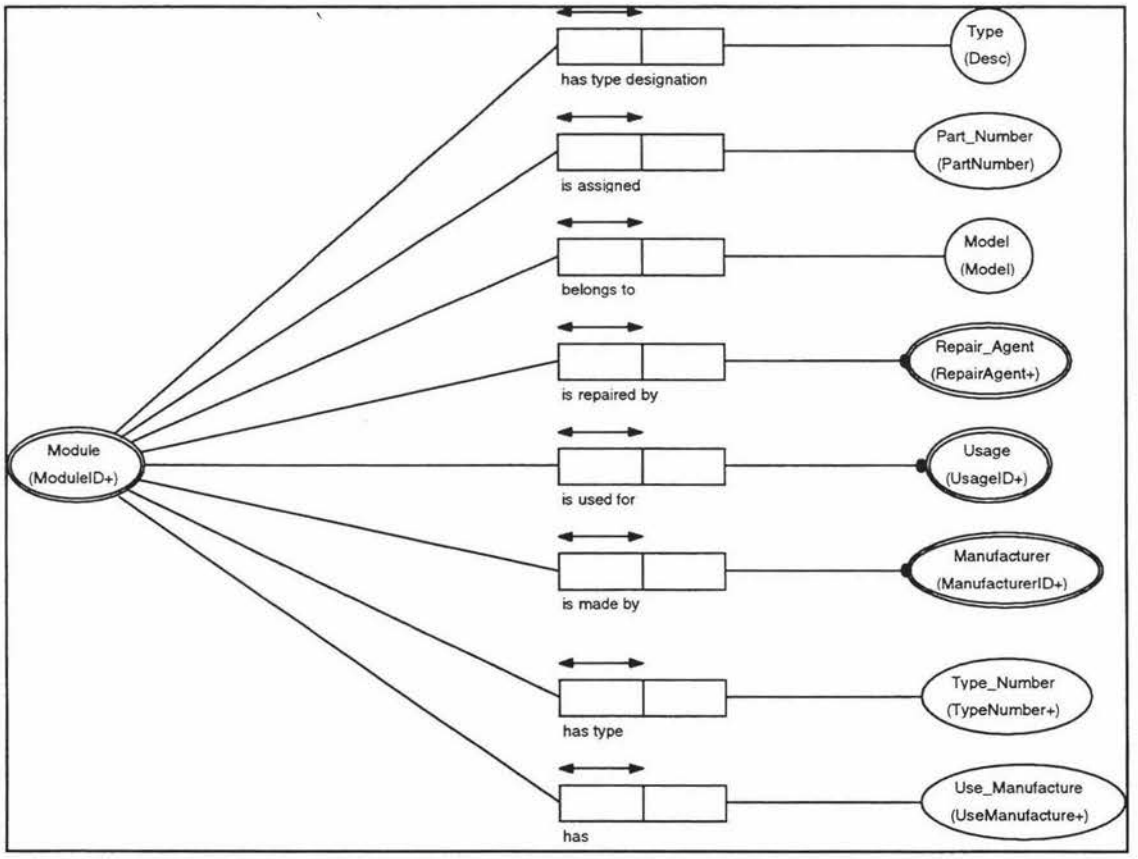


Figure 73. Module Descriptions Table.

Once again Figure 73 shows the addition of mandatory roles constraints on three object types.

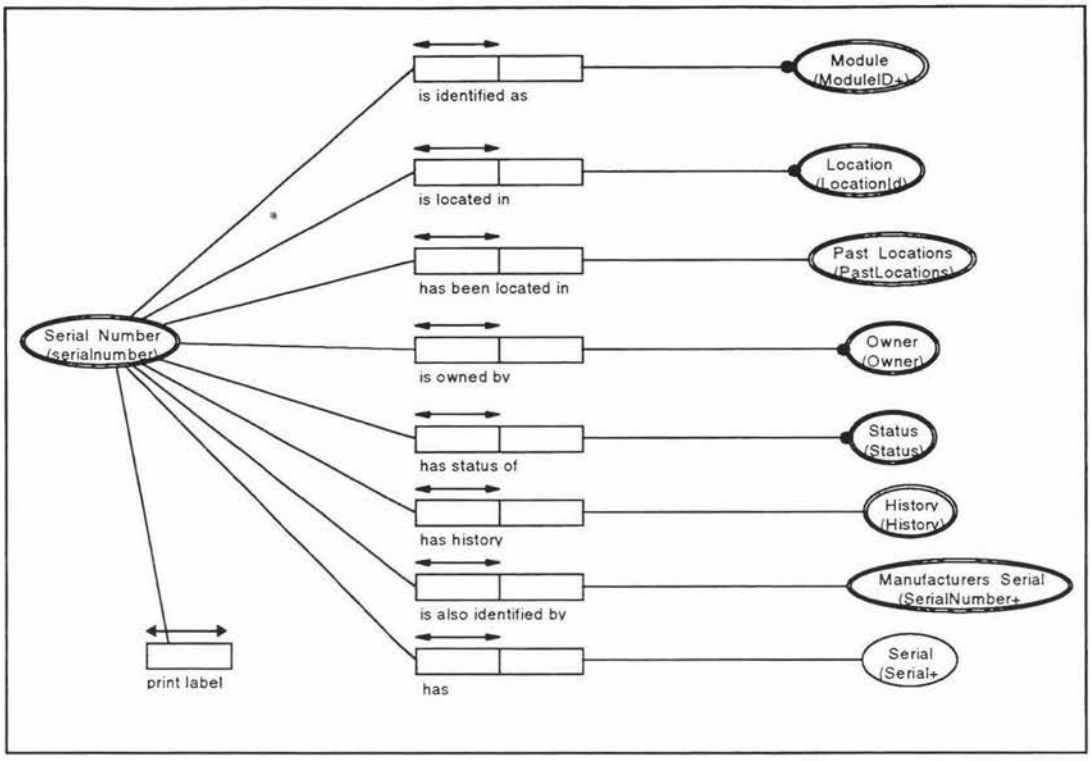


Figure 74. Modules Table.

Figure 74 once again shows the inclusion of the mandatory role on four of the fact types. The uniqueness constraints are unchanged from the previous version as this set was confirmed during the sentence verification process.

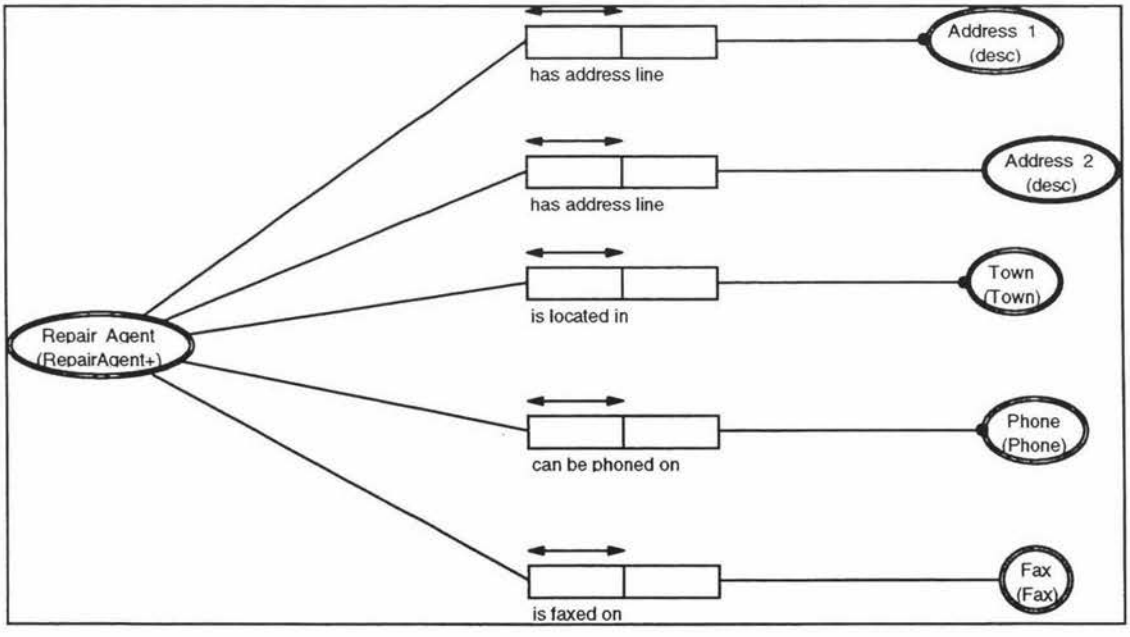


Figure 75. Repair Agents Table.

Again in Figure 75 the additional information incorporated is the inclusion of the mandatory role constraints.

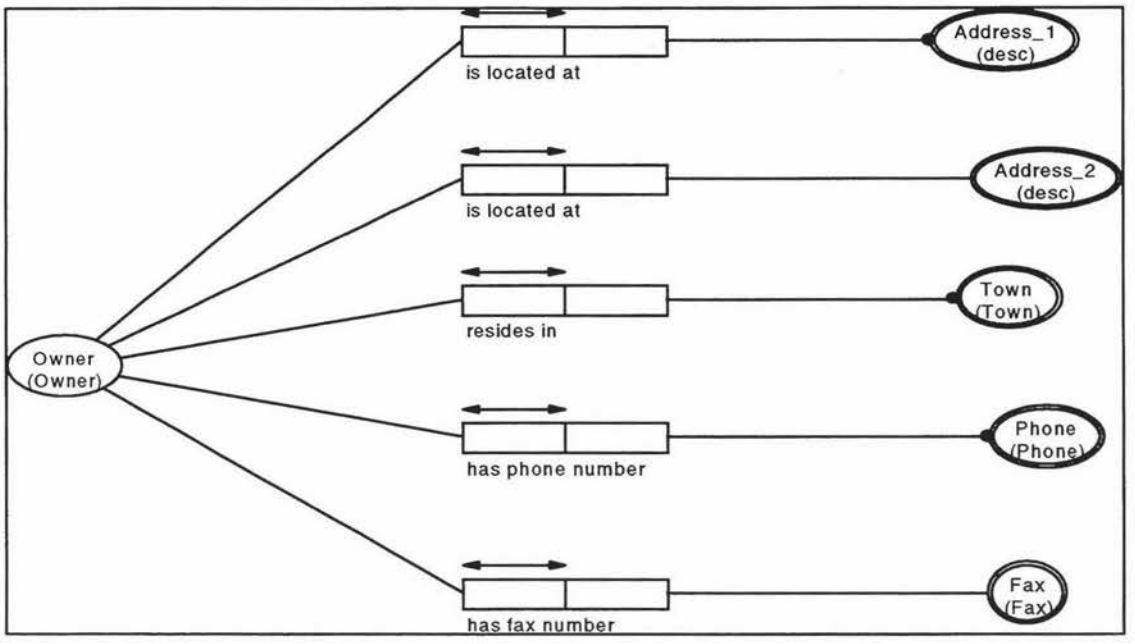


Figure 76. Owners Table.

The *Owners* table in Figure 76 is the same as shown in stage two except for the mandatory participation constraints on three object types.

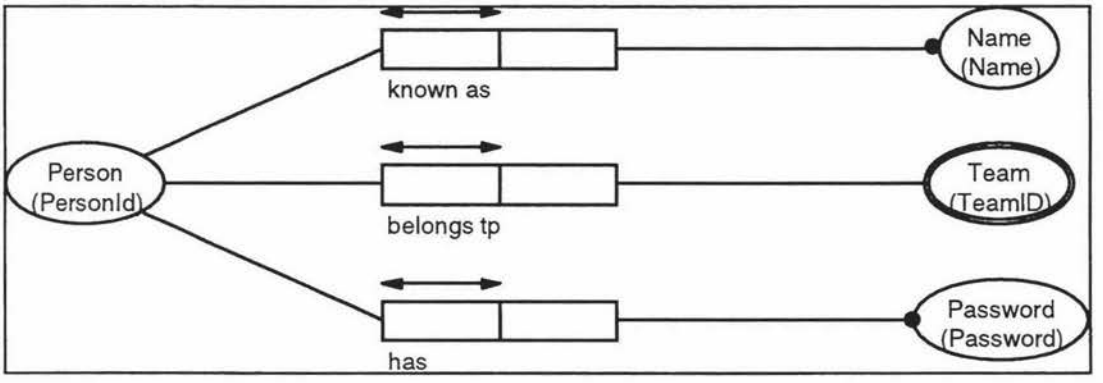


Figure 77. Person Table.

Figure 77 shows a changed version of the *Person* table from that shown in stage two. The verification process showed that rather than a quaternary fact type this table should consist of three binary fact types including two mandatory roles.

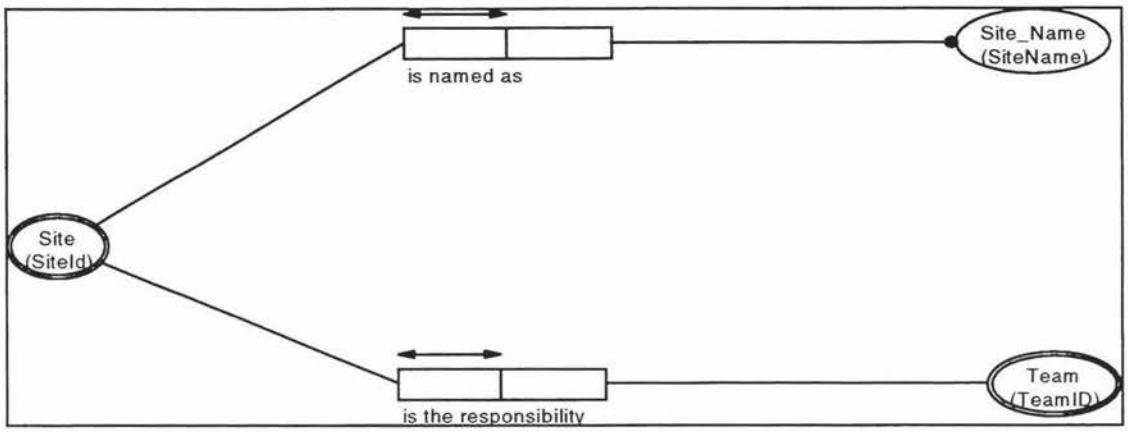


Figure 78. Sites Table.

Figure 78 shows the *Sites* table exhibiting the same characteristics as in stage two except for the mandatory role constraint on the *Site Name* role.

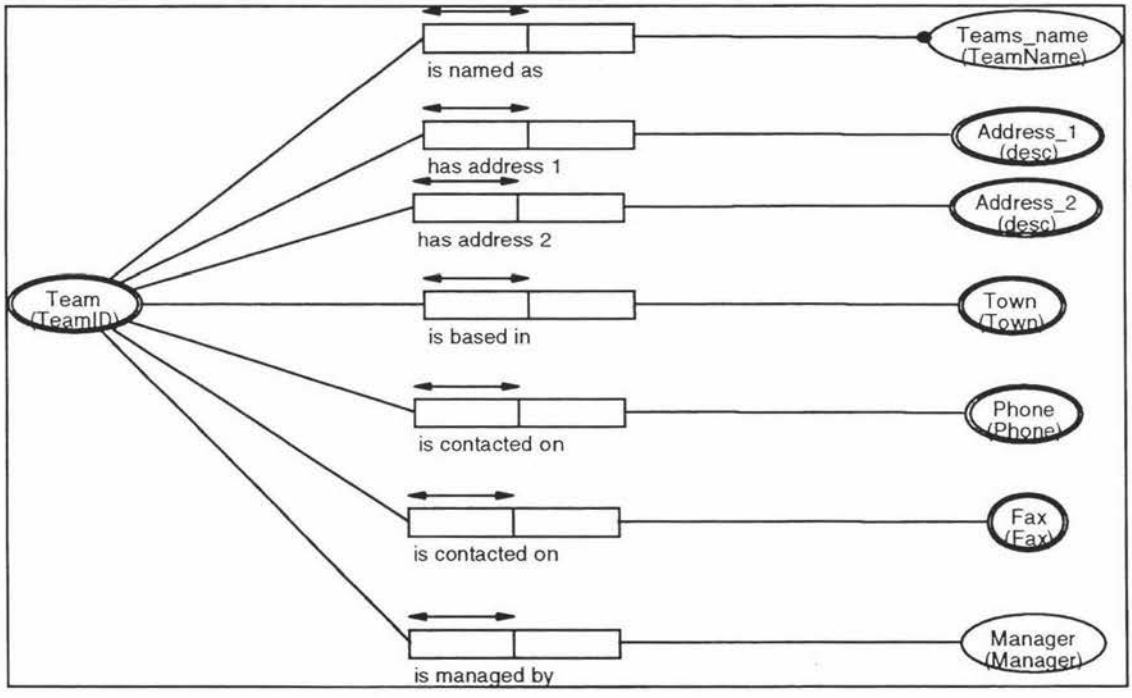


Figure 79. Teams Table

Once again the *Teams* table in Figure 79 is very similar to that in stage two except for the mandatory participation constraint in the involvement of the *Team Name* object in the fact type with *Team.*

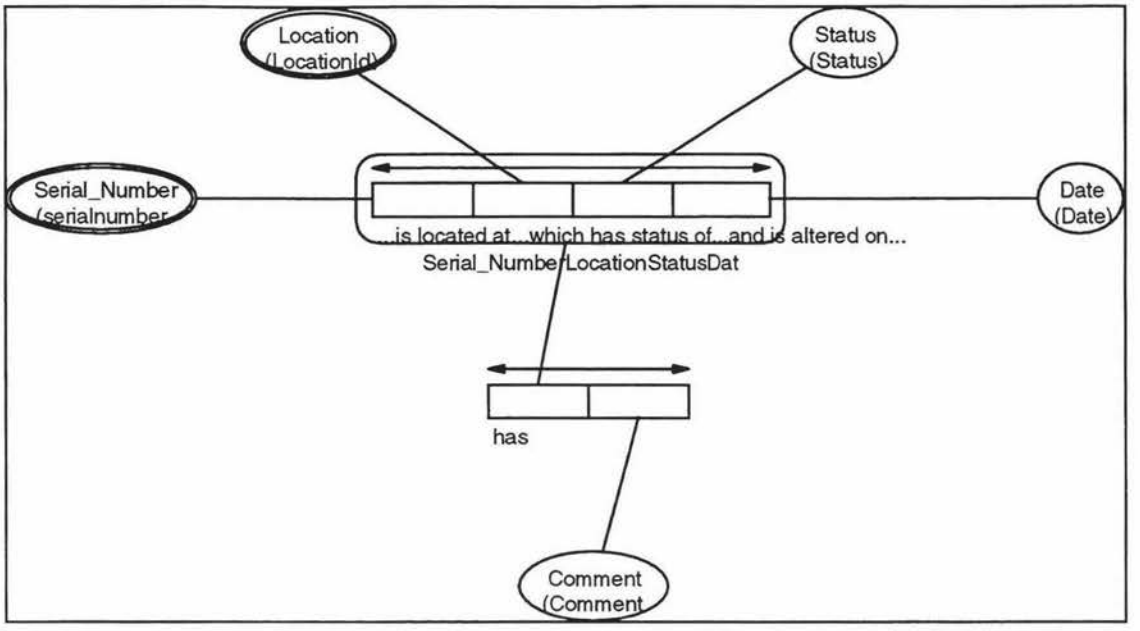


Figure 80.. Update Table.

The *Update* diagram in Figure 80 still involves an objectified quaternary fact type and a binary fact type.

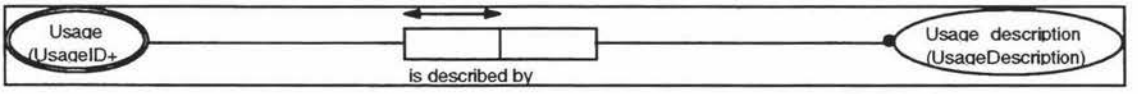


Figure 81. Usage Table.

Figure 81 shows the revised *Usage* diagram with a mandatory role being added to the participation of the *Usage_description* object in the fact type.

9.2.7 Stage 7 - Generation of new physical schema (relational).

Using the ONF algorithm the following set of tables is derived from the set of diagrams in the preceding stage. This schema is more normalised than the original design however it is likely that it could be further refined for implementation purposes. The table structures depicted in the following tables are mostly similar to the table structures derived from the original schema. However some have had changes in terms of their key structures and three of the original tables have been removed altogether. The revised table structures also contain information denoting which attributes are foreign keys. Table 36 through to Table 48 represent the revised database schema.

Attribute Name	*Type*	*Size*	*Not Null*	*Key*
Equip	Text	50	Y	P
Description	Text	50	Y	
Used_Usage	Number (Integer)		Y	

Table 36. Equipment Table Structure.

Attribute Name	*Type*	*Size*	*Not Null*	*Key*
Location	Text	50	Y	P
Equipment	Text	50	Y	F
Site	Text	50	Y	F
Used_Usage	Number(integer)		Y	F
Print_Label	Logical			

Table 37. Locations Table Structure

Attribute Name	*Type*	*Size*	*Not Null*	*Key*
Manufacturer	Number (Integer)	2	Y	P
Manf Name	Text	50	Y	

Table 38. Manufacturer's Table Structure.

Attribute Name	*Type*	*Size*	*Not Null*	*Key*
Module	Number (Integer)		Y	P
Type	Text	50		
Part Number	Text	50		
Model	Text	50		
Repair Agent	Text	50	Y	F
Used_Usage	Number (Integer)		Y	F
Manufacture_made	Number (Integer)		Y	F
Type_Number	Number (Integer)			
Manufacture_used	Number (Integer)			

Table 39. Module Table Structure.

Attribute Name	*Type*	*Size*	*Not Null*	*Key*
Owner	Text	50	Y	P
Address 1	Text	50	Y	
Address 2	Text	50		
Town	Text	50	Y	
Phone	Text	50	Y	
Fax	Text	50		

Table 40. Owners Table Structure.

Table 36 through to Table 40 are virtually the same as their counterpart tables in the original database design.

Attribute Name	Type	Size	Not Null	Key
ID	Text	3	Y	P
Name	Text	20	Y	
Password	Text	20	Y	
Team	Text	3		F

Table 41. Person Table Structure.

Table 41 is different to the original design in that it now has a single defined primary key rather than a set of attributes with no defined key. The table's attribute set is however still the same as in the original design.

Attribute Name	Type	Size	Not Null	Key
Repair Agent	Text	50	Y	P
Address 1	Text	50	Y	
Address 2	Text	50		
Town	Text	50	Y	
Phone	Text	50	Y	
Fax	Text	50		

Table 42. Repair Agent Table Structure.

Table 42 shows no changes from the original structure.

Attribute Name	Type	Size	Not Null	Key
Serial Number	Text	50	Y	P, F
Location	Text	50	Y	P, F
Status	Text	50	Y	P
Date	Date/Time	8	Y	P
Comment	Text	255		P

Table 43. Update Table Structure.

Table 43 contains information about what attributes make up the new key for this table. This has changed significantly from the original table design.

Attribute Name	Type	Size	Not Null	Key
Serial Number	Text	12	Y	P
Module	Number (Integer)		Y	F
Location	Text	50	Y	
Past_Locations	Text	255		
Owner	Text	50	Y	
Status	Text	50	Y	
History	Text	255		
Manufacturers serial	Text	50		
Serial	Number (Integer)			
Print_Label	Logical			

Table 44. Modules Table Structure.

Attribute Name	*Type*	*Size*	*Not Null*	*Key*
Site	Text	50	Y	P
Site Name	Text	50	Y	
Team	Text	3		F

Table 45. Sites Table Structure.

Attribute Name	*Type*	*Size*	*Not Null*	*Key*
Team	Text	3	Y	P
Teams Name	Text	50	Y	
Address 1	Text	50		
Address 2	Text	50		
Town	Text	50		
Phone	Text	50		
Fax	Text	50		
Manager	Text	50		

Table 46. Teams Table Structure.

Table 44 through to Table 46 are similar to their counterpart tables in the original database design.

Attribute Name	*Type*	*Size*	*Not Null*	*Key*
Team	Text	3	Y	P, F
Site	Text	50	Y	P, F
Repair Agent	Integer		Y	F

Table 47. Defaults Table Structure.

Table 47 has the same set of attributes as the original version of this table but an altered key structure with two attributes now combining to form a composite primary key.

Table 48 shows no changes from the original structure.

Attribute Name	*Type*	*Size*	*Not Null*	*Key*
Usage	Number (Integer)		Y	P
Usage description	Text	50	Y	

Table 48. Usage Table Structure.

The schema used for this experiment shows a number of changes as it has progressed through the schema redesign process. The original database definition contains 16 tables and 82 attributes with 14 of those tables having single attribute primary keys. In contrast the revised definition has 13 tables containing 66 attributes.

For the database schema used in this experiment the redesign procedure has helped to identify an area of the design that appears to serve no useful purpose. The remainder of the changes concentrate on alterations to the key structures for the tables. Due to the changes having minimal effect on the meta-data, incorporating data values from the original database schema should entail minimal data migration problems.

In contrast the next section shows a database design that has been derived for this same universe of discourse using a conventional NIAM analysis and design methodology.

9.3 Associate researcher's activities

To offer a degree of experimental control and a basis for comparison, an associate researcher interviewed the expert, and using NIAM techniques, designed a schema to fulfil the requirements of the same universe of discourse as discussed in the preceding section. What follows are the results of that process.

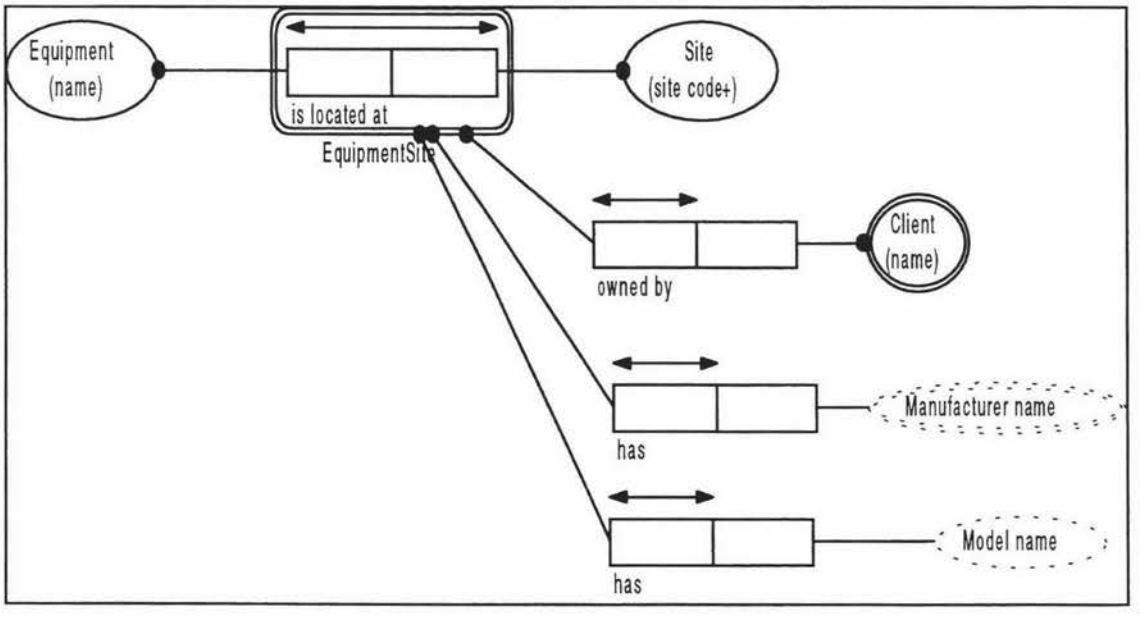


Figure 82. Equipment Diagram.

The diagram in Figure 82 combines information from parts of several diagrams shown in section six of the SRDP. Namely these are the equipment table in Figure 70, the locations table in Figure 71 and the manufacturers table in Figure 72.

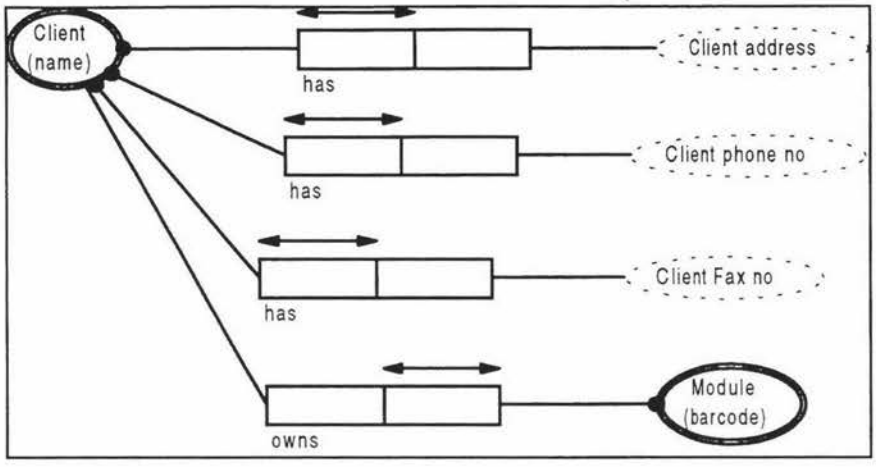


Figure 83. Client Diagram.

The diagram in Figure 83 also combines information from parts of several diagrams shown in section six of the SRDP. These are the modules table in Figure 74 and the owners table in Figure 76.

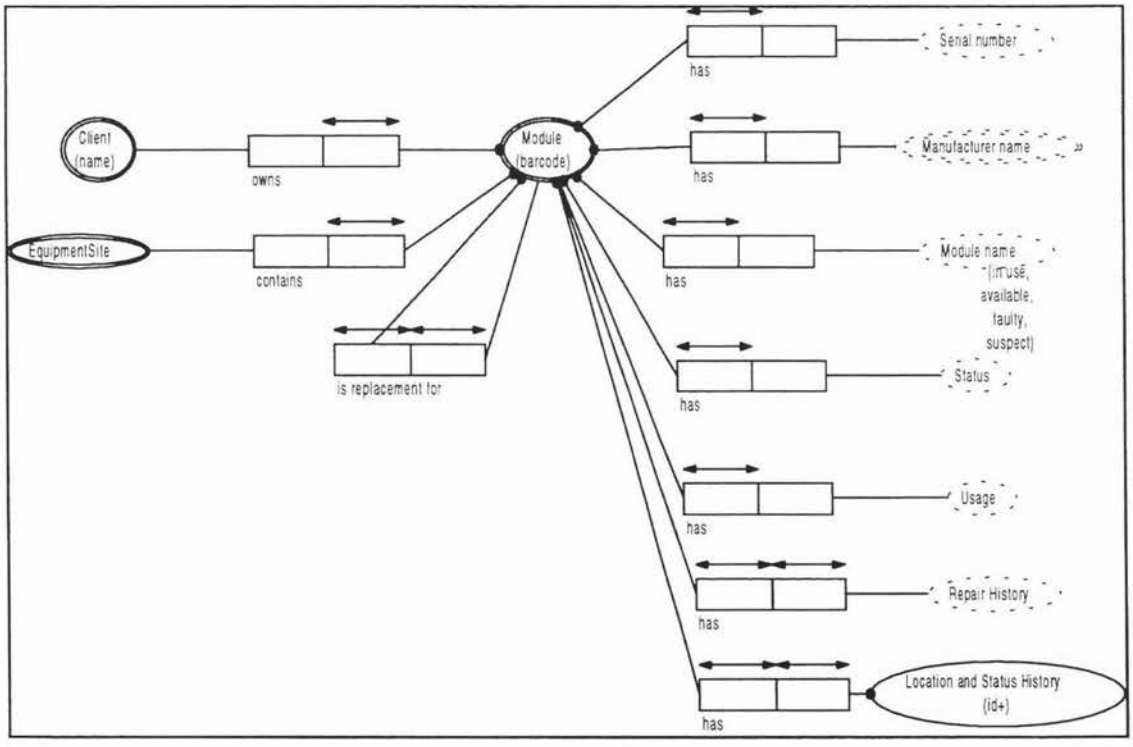


Figure 84. Module Diagram.

The diagram in Figure 84 is an agglomeration of several diagrams shown in section six of the SRDP. These are the modules table in Figure 74, the usage table in Figure 81, the site table in Figure 78 and the owners table in Figure 76.

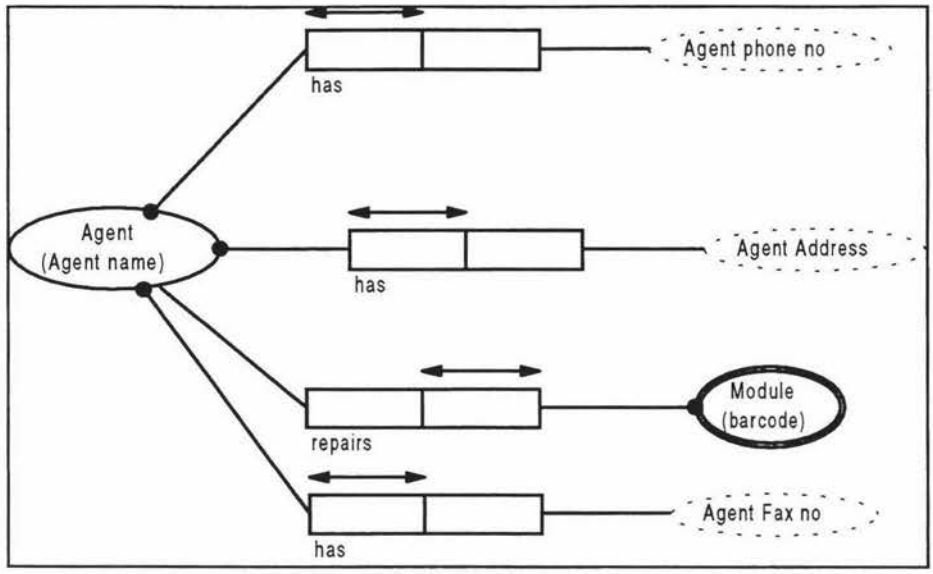


Figure 85. Agent Diagram.

The diagram in Figure 85 combines information from the repair agents table in Figure 75, and the module table in Figure 74.

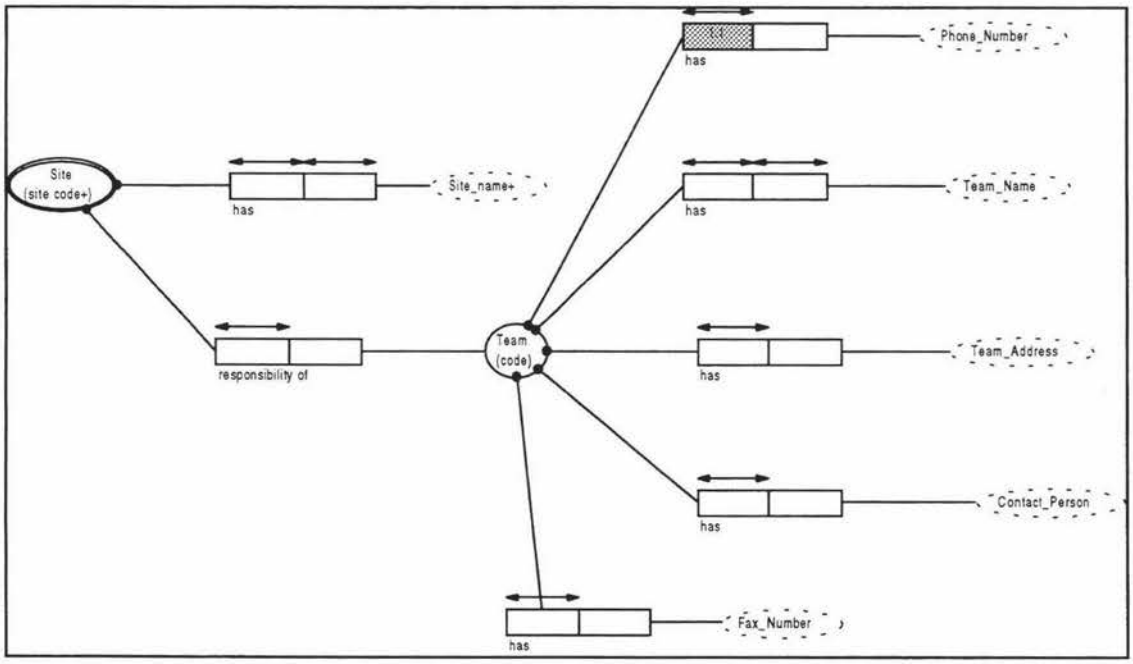


Figure 86. Site Diagram.

The diagram in Figure 86 also combines information from parts of several diagrams shown in section six of the SRDP. These are the team table in Figure 79 and the site table in Figure 78.

The six diagrams shown above translate to a six table database schema. The composition of all six tables is shown in the six figures below.

Attribute Name	*Type*	*Size*	*Key*
Agent	Text	20	P
Agent Address	Text	40	
Agent Phone Number	Text	10	
Agent Fax Number	Text	10	

Table 49. Agent table Structure..

Attribute Name	*Type*	*Size*	*Key*
Client	Text	20	P
Client Address	Text	30	
Client Phone Number	Text	10	
Client Fax Number	Text	10	

Table 50. Client Table Structure.

Attribute Name	*Type*	*Size*	*Key*
Equipment	Text	20	P
Located Site	Number		P, F
Owned Client	Text	20	F
Manufacturer Name	Text	20	
Model Name	Text	20	

Table 51. Equipment Table Structure.

Attribute Name	*Type*	*Size*	*Key*
Module	Text	12	P
Client	Text	20	F
Equipment	Text	20	F
Located Site	Number		F
Manufacturer Name	Text	20	
Module Name	Text	20	
Serial	Text	20	
Status	Logical	20	
Usage	Text	20	
Client0	Text	20	F
Repair History	Text		
Location / Status Hist	Number		
Agent	Text	20	F
Replacement Module	Text	12	F

Table 52. Module Table Structure.

Attribute Name	*Type*	*Size*	*Key*
Site	Number		P
Responsibility Team	Text	5	F
Site Name	Text	20	

Table 53. Site Table Structure.

Attribute Name	*Type*	*Size*	*Key*
Team	Text	5	P
Phone number	Text	10	
Fax Number	Text	10	
Contact Person	Text	15	
Team Address	Text	30	
Team Name	Text	25	

Table 54. Team Table Structure.

9.3.1 Comparison and discussion.

The three versions of schemata for this universe of discourse contain a number of differences. Although the numerical analysis in Table 55 gives an indicative view of the differences among the three schemata, an objective evaluation as to which better fulfils the requirements of the universe of discourse is more difficult to achieve. Such a qualitative assessment would need to take into account factors outside the scope of this research. There are however a number of relevant comments and clarifications concerning the numerical analysis. The first of these is that there is a significant decrease in the quantity of each object type from the original design to the revised design and then from the revised design to the associate researcher's design.

	Original design	*Revised design*	*Associate Researchers design*
Tables	16	13	6
Attributes	83	66	36
Diagrams	16	13	5
Objects	82	65	34
Fact types	62	36	29

Table 55. Numerical Comparison.

The decreasing quantitative trend is due to a number of contributing factors. Firstly the elimination of the three tables present in the original design containing no data and whose attributes were duplicated elsewhere in the schema accounts for most of the reduced quantities present in the revised design. Other differences between these two designs such as the reassignment of key structures on certain tables does not show up

in a numerical analysis. In addition to the tangible differences noticeable between these two designs there are intangible differences. The novice designer, having gone through the process will have had the opportunity to be more familiar with the reasons for having this particular design. Having verified the series of example sentences the novice designer should also have greater confidence that the design reflects their personal view of the universe of discourse.

The reasons for the difference in quantities between the revised design and the associate researcher's design are diverse. The underlying reason stems from the different starting points from which these designs were built. The associate researcher has designed a schema unencumbered by the assumptions present in any existing schema. The quantitative analysis indicates that the associate researcher's design is a more concise representation of the universe of discourse. This conciseness comes from reducing the quantity of attributes in the associate researcher's design that were apparent for functional reasons in the revised design. Without the need to carry these functional attributes the number of tables has decreased as indicated in Table 55. The reduced number of diagrams can also be attributed to the relatively unconstrained manner in which each diagram's boundary is decided. A relevant issue to be cognisant of when considering the comparison between these designs is that the associate researcher's design unlike the original design is yet to have application code placed against it. If this were to happen it is likely that there will be changes to the design probably requiring an increase in the number of attributes and tables.

Conclusions on what is a better or worse design are not necessarily valid if based solely on a quantitative analysis. In this particular case it appears that the design with the lesser number of attributes and tables depicts a more concise and more representative design. However it is possible in other cases that a design with fewer attributes and tables actually indicates that there are parts of the universe of discourse not depicted in that design.

Database schemata are usually intended to be implemented on live database management systems in real working environments. It is not until this has happened, and the system has been in operational use for some time, that an informed assessment of the system's quality can be made (Fertuck, 1992). This schema could have coding

attached to it at its current point of development. Were this to be done it could then be placed in an operational setting and from there be evaluated for the usefulness, appropriateness, adaptability and longevity of its design.

For the database schema used in this section, there already exists a rudimentary level of coding. The current coding is tightly coupled to the structure of the schema because the DBMS used has few ways for decoupling it. The most powerful way of doing this is by using dynamic views of tables and parts of tables (rather than the actual tables). The types of DBMSs used by novice designers either do not support these or only do so in a rudimentary fashion.

SECTION SEVEN

CONCLUSIONS

10 Conclusions

The database schemata used in the body of this report, in terms of the number of tables they contain, correlate to the approximate novice designer's database size identified through the survey. They can however be larger than this as verified by mention of the Forestry database in chapter three, that has in excess of sixty tables.

The survey phase of the research strongly supports the existence of novice designers within the community. While one objective of the research was to show this existence, it did not set out to suggest the proportion of people who might fall into this category. Although the research has produced statistics that appear to indicate a prevalence of novice designers it should be noted that the methods used to collect these figures make them unsuitable for extrapolation purposes.

Many of the novice designers included in this research indicated that they have built database applications, with little understanding of design principles. Whilst many of them pursued some avenue to improve their DBMS skills many respondents reported that their chosen avenue did not help them in understanding database design principles.

The research supported the idea that novice designers are likely to be found in many vocations, although those reported tend to be white collar workers with a fair degree of autonomy in their work place.

In terms of quality, the usefulness of the databases identified in this research has been subjectively assessed by the novice designers. All that can be reliably inferred from this is that these databases exist and are used at least to some extent. The schema designs, however are unlikely to be optimally configured.

The application of the SRDP is likely to produce a more optimal schema design. The extent of improvement will largely depend on the quality of the original design. In the worst case, the process of applying the procedure to the existing design will encourage the novice designer to think about the fit of aspects of their design with their universe

of discourse. The inducement of these thought processes may in time manifest themselves as design changes to improve the quality of the database application.

A speculative point brought to light from the survey is that there seems to be few easily accessible resources that talk in enough detail about schema design. Consequently it is possible that in order for novice designers to produce an effective design at present it is likely that they will do so through trial and error.

The resultant schema design derived from undergoing the SRDP is unlikely to be radically different from the original schema. However there are likely to be numerous subtle differences which, combined with any insights the novice designer gains from implementing the procedure, will result in improvements over the original schema. The magnitude of those improvements will, to a large extent, be dependent on the degree of quality present in the original design. Although this means that it's unlikely to be an ideal design, it does follow that data migration from the original design to the revised design may not be a too complex task. Conversely, designing a new schema from a conventional analysis of the universe of discourse might result in major problems when it comes time to move the existing data to the new structure.

The schema derived from a conventional application of the NIAM CSDP should produce a highly normalised schema. This schema is likely to be more concise and normalised than that derived from the SRDP.

10.1 Recommendations for further research

Before a schema re-design process can be effectively utilised on Σ databases being used operationally there needs to be a method or some guidelines to assist the novice designer in migrating their existing data from their original design to the revised design. Although the schema design and application code represent an investment in time and effort from the novice designer, it is the data contained in the database that is likely to be of more value and may even be critical to the organisation's viability.

Having established that novice designers exist in the community, it would be interesting to establish in what sort of numbers they are present. A quantitative assessment would not be a simple task to accomplish because of the diverse disciplines

in which novice designers may be found and because they might be hard to uncover unless they had recently or were in the process of building a database. To examine this question it may be desirable to stratify the problem by narrowing an investigation to particular sections of the community. Such sections may include selected industry sectors, non-profit organisations or hobbyists.

The present research has not attempted to investigate how changes to the universe of discourse over time will affect databases, designed by novice designers. To do this would require a longitudinal study of a sample number of candidate schemata.

The experiment section of the present report is limited partly because it only examines a single database design. Therefore the issues raised from applying the SRDP to it are also particular to that design. Application of the SRDP across a number of Σ designs may raise more issues and give a stronger picture of what types of design weaknesses the SRDP reveals. It is interesting to note that the latest version of Infomodeler (V2) provides the facility for reverse engineering which may facilitate the use of the SRDP.

To be most useful to novice designers the SRDP needs to be delivered in a form that requires minimal effort and understanding on the part of the novice designer. One effective method of delivery is to incorporate it into a computerised CASE-like tool. This will allow the user to receive prompts for the information the tool needs to complete the process rather than the user being required to drive the process.

To minimise potential bias in the application of the SRDP it would ideally be implemented by novice designers. However without incorporation into a CASE-like tool the procedure is too complex therefore research could be conducted by a database professional assisting a novice designer through the process.

SECTION EIGHT

APPENDICES

Appendix One

Survey Results

Number of questionnaires sent: 108

Number of responses: 40

Number of valid responses: 28

Section One.

Question 1. Number of people employed at work-places.

Option	1 - 4	5 - 10	11 - 20	21 - 50	50+
# Respondents	3	3	1	4	16

Question 2. Job titles.

Accountant	Environmental Officer	Museum Director
Admin Superintendent	Forest Management	Personal Assistant
Analyst	Health & Safety	Parks Asset Management
Assistant Treasurer	Help Desk	Programmer
Corporate Accountant	Intelligence Analyst	Reference Librarian
Development Executive	Information Dev Officer	Systems Administrator
Director	Library Manager	Systems Manager
Electronics Technician	Market Analyst	Team Leader

Question 3. Percent of work time using computers.

Option	0 - 10%	11 - 40%	41 - 60%	61 - 90%	91 - 100%
# Respondents	0	7	2	14	5

Question 4. Computer skills rating.

Option	None	Poor	Moderate	Good	Very Good
# Respondents	0	0	3	16	9

Question 5. Regularly used software packages.

Option	# Respondents
Word Processor	26
Spreadsheet	26
DBMS	24
Desktop Publish	4
Communication	9
E-Mail	16
Treasury System	1

Option	# Respondents
Project Mgmt	1
Graphics	1
Time Series	1
GIS	1
Visual basic	1
CAD	1

Question 6. Database packages used.

Option	# Respondents
Paradox	5
Paradox Windows	6
DBase	4
Microsoft Access	17
Advance Revelation	2
FoxPro	0

Option	# Respondents
Q&A	1
Informix	2
Pick	1
Clarion	1
Image II	1
SQL Server	1

Question 7. Length of time spent using a DBMS (years).

Average	3.267857
Median	3
Standard deviation	1.99296

Question 8. Number of databases built.

Average	4.019231
Median	2.5
Standard deviation	3.694539

Question 9. 27 respondents indicated they had designed a database in the last three years, 1 said they had not.

Section Two

Question 1. DBMS used for most recent database development.

Option	# Respondents
Paradox	4
Paradox Windows	2
DBase	0
Microsoft Access	15
Advance Revelation	1

Option	# Respondents
FoxPro	0
Q&A	1
Informix	1
Pick	1

Question 2. Number of tables in database.

Option	1	2 - 5	6 - 10	11 - 20	21 - 40	41+
# Respondents	0	5	6	10	1	2

Question 3. Number of rows in the largest tables.

Option	1 - 10	11 - 50	51 -200	201-1000	1000-10,000	10,000+
# Respondents	2	1	3	6	7	6

Question 4. Database's regularity of use.

Option	Constant	Hourly	Daily	Weekly	Monthly	Annually	Intermittent
# Respondents	7	5	5	5	4	0	0

Question 5. Proportion of employees that use the database.

Option	No-one	Just one	A few	Most people	Everyone
# Respondents	1	2	19	2	1

Question 6. The database's primary function.

Option	Time keep	Records	Billing	Diary	Data Analysis
# Respondents	1	17	2	2	2

Section Three

Question 1. Understanding of database design issues.

Option	None	A little	Moderate	Somewhat	Very Good
# Respondents	0	6	7	9	5

Question 2. Database courses attended.

Option	None	Polytech	Product	Industry	University	Night	Other
# Respondents	8	2	14	7	6	0	5

Question 3. How much database design help acquired from course.

Option	Not at all	A little	Moderate	Somewhat	A lot
# Respondents	2	6	3	3	3

Question 4. Method of table selection for recent database.

Option	Intuition	Manual	Formal process	On-line help	Trial & error	Modification	Expert advice	User input
# Respondents	8	8	7	0	6	6	7	2

Question 5. Manual's helpfulness with design issues

Option	Not at all	A little	Moderate	Somewhat	Extreme
# Respondents	3	11	6	3	1

Question 6. Similarity of past table structure to present.

Option	Not at all	A little	Moderate	Somewhat	Very
# Respondents	0	2	5	11	7

Question 8. To what degree have structural changes had on-going effects.

Option	None	A little	Moderate	Somewhat	A lot
# Respondents	7	7	5	2	0

Appendix Two *Sample Questionnaire*

End-user Database Questionnaire

My name is Mike Ryder and currently I am undertaking research in the Information Systems Department at Massey University. My thesis topic concerns the area of database usage, in particular resolving problems met by end-users using relational database management systems such as Paradox, DBase, Access and others.

Criteria

There are certain criteria respondents need to fulfil to usefully complete this questionnaire. If you can answer yes to the following questions, then please continue on and complete the questionnaire. If not, then please pass it on to someone else who may fill these criteria.

A. Do you use a relational database management system?

B. Have you built tables or an application with a relational database management system?

C. Do you consider yourself **other than** an Information Systems professional?

Survey questions are focussed on identifying your experiences with the use of database systems and in particular problems concerning the design of database schemas. Results of the survey will be used to define practical strategies available to database designers such as yourself that will assist in re-designing schemas, or helping to recover existing schemas, that have out lived their usefulness.

A follow-up to the survey will take place at a later date by interviewing a small number of willing respondents on a one-to-one basis to investigate what strategies they used to overcome difficulties encountered during design. This interview will be arranged so as to cause minimal inconvenience to the interviewee. The results obtained will be used to propose appropriate strategies for the purpose of re-engineering database designs to better suit user needs.

This questionnaire is designed for research purposes only and any information revealed herein will remain anonymous. Dissemination of results will only take place in a summarised, aggregated form. Once completed, please use the self addressed envelope to return the questionnaire to myself:

Mike Ryder
C/- Information Systems Department
Massey University
Private Bag 11-222
Palmerston North

Alternatively you can fax it to me at 06-350-5611.

Thank you for your cooperation

INSTRUCTIONS

If extra room is required for any questions or you wish to make any comments, please feel free to attach extra sheets or use the reverse side of the questionnaire. Please ensure that any extra sheets are clearly marked to indicate the question to which they pertain. Thank you.

SECTION ONE:

Database involvement:

In this section I am trying to draw out the scope of your exposure and experience with database management systems on micro-computers.

1. How many people are employed at your work place? (Please tick one.)

   A ☐1 - 4 B ☐5 - 10 C ☐ 11 - 20 D ☐ 21 - 50 E ☐ 50+

2. What is your role at work? _____

3. What percentage of your job is spent interacting with computers?

   A ☐ 0 - 10% B ☐11 - 40% C ☐41 - 60% D ☐ 61 - 90% E ☐91 - 100%

4. How do you rate your computer skills? (Please tick one.)

   A ☐ None B ☐ Poor C ☐ Moderate D ☐ Good E ☐ Very good

5. What packages have you used or do you regularly use? (Please tick.)

   A ☐Word processor B ☐Database Management System C ☐Communications
   D ☐Spreadsheet E ☐Desktop Publisher F ☐E-Mail
   G ☐Other (please specify)_____.

6. What database packages do you use? (Please tick one or more.)

A ☐Paradox B ☐Microsoft Access
C ☐Paradox for windows D ☐Advanced revelation
E ☐DBase F ☐FoxPro
G ☐Other (please specify) _____.

7. How long have you been using a database package? _____

8. How many databases have you designed and/or built? _____

9. Have you designed a database in the last three years? (Please circle) A Yes / B No.

10. Comments:

_____

_____

SECTION TWO:

Products used

In this section you are asked to give more details of the most significant database you have designed. If you have designed a number of databases choose a recent one that has a more complex structure (a higher number of tables may be a useful rule of thumb).

1. What package was used in the development of the most recent, significant database?

A☐ Paradox B☐ Microsoft Access
C☐ Paradox for windows D☐ Advanced revelation
E☐ DBase F☐ FoxPro
G☐ Other (please specify) _____

2. How many tables did this database consist of?

A ☐One B ☐2 - 5 C ☐6 - 10 D ☐11 - 20 E ☐21 - 40 F ☐41+

3. Approximately how many rows are there in the largest tables?

A☐1-10 B☐11-50 C☐51-200 D☐201-1,000 E☐1,000-10,000 F☐10,000+

4. How often is this database now used?

A☐ Constantly B☐ Hourly C☐ Daily D☐ Weekly
E☐ Monthly F☐ Annually G☐ Intermittently H☐ Never

5. How many people in your organisation use, or used to use, this database?

A☐No-one B☐Just one C☐A few D☐Most people E☐Everyone

6. What is the primary function of the database?

A☐ Timekeeping B☐ Record-keeping C☐ Billing
D☐ Other (please specify) _____.

7. Comments:

_____

_____

_____

SECTION THREE

Problems encountered

In this section you will be asked about your understanding of database design and problems you may have encountered during the design process.

1. How do you rate your understanding of the issues involved in designing a database? (Please tick only one.)

A ☐ None B ☐ A little C ☐ Moderate D ☐ Somewhat E ☐ Very good

2. What form of database courses have you attended? (Please tick those appropriate.)

A ☐ None B ☐ Product C ☐ University course
D ☐ Polytechnic course E ☐ Industry F ☐Community education class
G ☐ Other (please specify) _____.

3. How much do you feel the course helped you to decide what tables your database needed? (Please tick only one.)

A ☐ Not at all B ☐ A little C ☐ Moderately D ☐ Somewhat E ☐ A lot

4. How did you choose the tables your database needed? (Please tick those appropriate.)

A ☐ Intuition B ☐ Formal process C ☐ Trial & error D ☐ Expert advice
E ☐ Manual F ☐ Using on-line help G ☐ Modification of existing tables
H ☐ Other (please specify) _____.

5. How helpful did you find the manuals and documentation that came with the DBMS for deciding which tables your database needed? (Please tick only one.)

A ☐ Not at all B ☐ A little C ☐ Moderately D ☐ Somewhat E ☐ Extremely

6. How similar are the table structures in the most recent version compared with when you started? (Tick only one.)

A ☐ Not at all B ☐ A little C ☐ Moderately D ☐ Somewhat E ☐ Very

7. If the table structures have changed then why have these changes occurred?

_____

_____

8. To what degree have these structural changes resulted in further problems, possibly requiring further changes in table structures? (Please tick only one.)

A ☐ None B ☐ A little C ☐ Moderate D ☐ Somewhat ☐ A lot

9. Comments:

_____

_____

_____

☐ Please tick here if you are willing to be interviewed at a later date and include your name and contact details:

Name: _____

Address: _____

Contact No: _____

Your contribution to this survey is greatly appreciated. **Thank you**

Appendix Three *Frame-work for case study interviews*

Section One - Background to project (to enlighten on situation context)

1. Why choose to develop own database?

2. How was the project's scope arrived at?

3. Where did the impetus for the project come from? (DBMS or situation)

4. Did you use forms and reports as design aids?

Section Two - The database

1. Can you draw a database structure? (Do you have a clear mental picture of it?)

2. Has the number of tables increased since initial design?

3. Have you come across inconsistent data in answers and queries

4. Do you have or know of any:
    - data duplication in your system
    - derived data in your system
    - redundant data in your system

5. How long do you think this system will remain useful?

6. What would happen if there was a sudden fundamental change in the data?

Section Three - System adaptability

1. Does the system achieve the purpose for which it was built?

2. Are there any changes you'd like to implement?

3. Why haven't they been implemented?

4. What will happen when data requirements or functions change? (Adapt or throw away)

5. How would you assess the impact analysis of altering the schema?

6. Would a set of sentences help?

7. Would a diagram help?

8. How much has this experience helped you for designing other databases?

Appendix Four *Example sentences and object definitions*

Account Table

Objects:

---

1. Account: Nested object-type
2. Account_Description: Simple object-type identified by 'AccountDescription' and implemented as 'varchar 30'
3. Account_Number: Simple object-type identified by 'AccountNumber' and implemented as 'varchar 5'
4. Account_Type: Simple object-type identified by 'AccountType' and implemented as 'varchar 8'
5. Old_Number: Simple object-type identified by 'OldNumber' and implemented as 'varchar 45'
6. Sub_Account: Simple object-type identified by 'SubAccountNumber' and implemented as 'varchar 5'
7. Sub_Account_Description: Simple object-type identified by 'SubAccountDescription' and implemented as 'varchar 12'

Facts:

---

1. Account has Account_Type
   Every Account has exactly one Account_Type

Examples:

Account '02020,FORST' has Account_Type 'Income'
Account '02020,FORST' has Account_Type 'Expense'
Account '02020,NZDB' has Account_Type 'Income'

2. Account has Old_Number / Old_Number equates to Account
   Every Account has exactly one Old_Number

Examples:

Account '02020,FORST' has Old_Number '0010.0202'
Account '02020,FORST' has Old_Number '0011.0202'
Account '02030,NZDB' has Old_Number '0010.0202'

3. Account_Description describes Account / Account has Account_Description
   Every Account has exactly one Account_Description

Examples:

Account_Description 'Contracts Research' describes Account '02020,FORST'
Account_Description 'Minor Contracts' describes Account '02020,FORST'
Account_Description 'Contracts Research' describes Account '02030,NZDB'

4. Account_Number contains Sub_Account / Sub_Account is associated with Account_Number
   Every Account_Number contains at least one Sub_Account and
   Every Sub_Account is associated with at least one Account_Number

Examples:

Account_Number '02020' contains Sub_Account 'FORST'
Account_Number '02020' contains Sub_Account 'NZDB'
Account_Number '02030' contains Sub_Account 'FORST'
Account_Number '02030' contains Sub_Account 'NZDB'

5. Sub_Account_Description has Account / Account describes Sub_Account_Description
   Every Account describes exactly one Sub_Account_Description

Examples:

Sub_Account_Description 'FoRST' has Account '02020,FORST'
Sub_Account_Description 'NZDB/NZDRI' has Account '02020,FORST'
Sub_Account_Description 'FoRST' has Account '02030,NZDB'

Credits Table

Objects:

---

1. Account: Nested object-type
2. Amount: Simple object-type identified by 'Amount' and implemented as 'money (15,15)'
3. Client: Simple object-type identified by 'ClientID' and implemented as 'integer'
4. Credit: Nested object-type
5. Dept: Simple object-type identified by 'Dept' and implemented as 'smallint'
6. General_Ledger: Simple object-type identified by 'GeneralLedgerNumber' and implemented as 'smallint'
7. Invoice: Simple object-type identified by 'InvoiceNumber' and implemented as 'integer'
8. Invoice_Date: Simple object-type identified by 'InvoiceDate' and implemented as 'date'
9. Month: Simple object-type identified by 'Month' and implemented as 'smallint'
10. Project: Simple object-type identified by 'ProjectID' and implemented as 'varchar 8'
11. Signal: Simple object-type identified by 'Signal' and implemented as 'varchar 8'
12. Year: Simple object-type identified by 'Year' and implemented as 'smallint'

Facts:

---

1. Client is assigned to Credit / Credit receives Client
   Every Credit receives exactly one Client

Examples:

   Client '25' is assigned to Credit '02020,FORST,23'
   Client '30' is assigned to Credit '02020,FORST,23'
   Client '25' is assigned to Credit '02020,FORST,66'

2. Credit belongs to Dept
   Every Credit belongs to exactly one Dept

Examples:

   Credit '02020,FORST,23' belongs to Dept '16685'
   Credit '02020,FORST,23' belongs to Dept '17002'
   Credit '02020,FORST,66' belongs to Dept '16685'

3. Credit has credit note Signal
   Every Credit has credit note exactly one Signal

Examples:

   Credit '02020,FORST,23' has credit note Signal 'CREDIT'
   Credit '02020,FORST,23' has credit note Signal 'DEBIT'
   Credit '02020,FORST,66' has credit note Signal 'CREDIT'

4. Credit has invoice amount Amount
   Every Credit has invoice amount exactly one Amount

Examples:

   Credit '02020,FORST,23' has invoice amount Amount '25000'
   Credit '02020,FORST,23' has invoice amount Amount '30000'
   Credit '02020,FORST,66' has invoice amount Amount '25000'

5. Credit is created in Month
   Every Credit is created in exactly one Month

Examples:

   Credit '02020,FORST,23' is created in Month '6'
   Credit '02020,FORST,23' is created in Month '12'
   Credit '02020,FORST,66' is created in Month '6'

6. Credit is created in Year
   Every Credit is created in exactly one Year

Examples:

   Credit '02020,FORST,23' is created in Year '1994'
   Credit '02020,FORST,23' is created in Year '1995'
   Credit '02020,FORST,66' is created in Year '1994'

7. General_Ledger is assigned to Account
    Every General_Ledger is assigned to at least one Account and
    Every Account has at least one General_Ledger that is assigned to it
Examples:
    General_Ledger '23' is assigned to Account '02020,FORST'
    General_Ledger '23' is assigned to Account '02020,NZDB'
    General_Ledger '24' is assigned to Account '02020,FORST'

8. Invoice refers to Credit / Credit has Invoice
    Each Credit has at most one Invoice
Examples:
    Invoice '1056' refers to Credit '02020,FORST,23'
    Invoice '1164' refers to Credit '02020,FORST,23'
    Invoice '1056' refers to Credit '02020,FORST,66'

9. Invoice_Date is recorded for Credit / Credit is assigned Invoice_Date
    Each Credit is assigned at most one Invoice_Date
Examples:
    Invoice_Date '6.6.94' is recorded for Credit '02020,FORST,23'
    Invoice_Date '7.7.94' is recorded for Credit '02020,FORST,23'
    Invoice_Date '6.6.94' is recorded for Credit '02020,FORST,66'

10. Project has Credit / Credit is assigned to Project
    Each Credit is assigned to at most one Project
Examples:
    Project '93-34' has Credit '02020,FORST,23'
    Project '93-66' has Credit '02020,FORST,23'
    Project '93-34' has Credit '02020,FORST,66'

Project Expense Table

Objects:

---

1. Date: Simple object-type identified by 'Date' and implemented as 'date'
2. Expense: Simple object-type identified by 'ExpenseCode' and implemented as 'smallint'
3. Expense_Amount: Simple object-type identified by 'ExpenseAmount' and implemented as 'money (15,15)'
4. General_Ledger: Simple object-type identified by 'GeneralLedgerNumber' and implemented as 'smallint'
5. Project: Simple object-type identified by 'ProjectID' and implemented as 'varchar 8'
6. ProjExp: Nested object-type
7. User_Desc: Simple object-type identified by 'UserDesc' and implemented as 'varchar 15'
8. Week_Ending_Date: Simple object-type identified by 'WeekEndingDate' and implemented as 'date'

Facts:

---

1. Expense_Amount quantifies ProjExp / ProjExp has Expense_Amount
    Every ProjExp has exactly one Expense_Amount
Examples:
    Expense_Amount '12.90' quantifies ProjExp '93-34,46,21.11.94,Photocopying'
    Expense_Amount '150.00' quantifies ProjExp '93-34,46,21.11.94,Photocopying'
    Expense_Amount '12.90' quantifies ProjExp '93-34,46,21.11.94,Vouchers'

2. General_Ledger refers to ProjExp / ProjExp has General_Ledger
    Each ProjExp has at most one General_Ledger
Examples:
    General_Ledger '100' refers to ProjExp '93-34,46,21.11.94,Photocopying'
    General_Ledger '250' refers to ProjExp '93-34,46,21.11.94,Photocopying'
    General_Ledger '100' refers to ProjExp '93-34,46,21.11.94,Vouchers'

3. Project with Expense on Date described by User_Desc
    Every Project must participate in this relationship
    Every Expense must participate in this relationship
    Every Date must participate in this relationship
    Every User_Desc must participate in this relationship
    Each Project, Expense, Date, User_Desc combination is unique
Examples:
    Project '93-34' with Expense '46' on Date '21.11.94' described by User_Desc 'Photocopying'
    Project '93-34' with Expense '46' on Date '21.11.94' described by User_Desc 'Vouchers'
    Project '93-34' with Expense '46' on Date '10.5.94' described by User_Desc 'Photocopying'
    Project '93-34' with Expense '52' on Date '21.11.94' described by User_Desc 'Photocopying'
    Project '93-66' with Expense '46' on Date '21.11.94' described by User_Desc 'Photocopying'

4. ProjExp has Week_Ending_Date
    Every ProjExp has exactly one Week_Ending_Date
Examples:
    ProjExp '93-34,46,21.11.94,Photocopying' has Week_Ending_Date '25.11.94'
    ProjExp '93-34,46,21.11.94,Photocopying' has Week_Ending_Date '12.05.95'
    ProjExp '93-34,46,21.11.94,Vouchers' has Week_Ending_Date '25.11.94'

5. ProjExp is chargeable
Examples:
    ProjExp '93-34,46,21.11.94,Photocopying' is chargeable
    ProjExp '93-34,46,21.11.94,Vouchers' is chargeable

Project Table

Objects:

---

1. Amount: Simple object-type identified by 'Amount' and implemented as 'money (15,15)'
2. Client: Simple object-type identified by 'ClientID' and implemented as 'integer'
3. Date: Simple object-type identified by 'Date' and implemented as 'date'
4. Group: Simple object-type identified by 'Groupcode' and implemented as 'integer'
5. Name: Simple object-type identified by 'Name' and implemented as 'integer'
6. Number: Simple object-type identified by 'number' and implemented as 'integer'
7. Project: Simple object-type identified by 'ProjectID' and implemented as 'varchar 8'
8. Staff: Simple object-type identified by 'Staffnumber' and implemented as 'integer'

Facts:

---

1. Project belongs to Group
    Every Project belongs to exactly one Group
Examples:
    Project '93-34' belongs to Group '1'
    Project '93-34' belongs to Group '2'
    Project '93-66' belongs to Group '1'

2. Project has agreed budget Amount
    Each Project has agreed budget at most one Amount
Examples:
    Project '93-34' has agreed budget Amount '25000'
    Project '93-34' has agreed budget Amount '30000'
    Project '93-66' has agreed budget Amount '25000'

3. Project has deposit Amount
    Each Project has deposit at most one Amount
Examples:
    Project '93-34' has deposit Amount '2500'
    Project '93-34' has deposit Amount '3000'
    Project '93-66' has deposit Amount '2500'

4. Project has expected end date Date
Each Project has expected end date at most one Date
Examples:
Project '93-34' has expected end date Date '6.6.94'
Project '93-34' has expected end date Date '7.7.94'
Project '93-66' has expected end date Date '6.6.94'

5. Project has inception date Date
Each Project has inception date at most one Date
Examples:
Project '93-34' has inception date Date '8.1.94'
Project '93-34' has inception date Date '22.2.94'
Project '93-66' has inception date Date '8.1.94'

6. Project has income received Amount
Each Project has income received at most one Amount
Examples:
Project '93-34' has income received Amount '25000'
Project '93-66' has income received Amount '25000'
Project '93-34' has income received Amount '30000'

7. Project has monthly accrual Amount
Each Project has monthly accrual at most one Amount
Examples:
Project '93-34' has monthly accrual Amount '1500'
Project '93-34' has monthly accrual Amount '1850'
Project '93-66' has monthly accrual Amount '1500'

8. Project has overhead of Amount
Each Project has overhead of at most one Amount
Examples:
Project '93-34' has overhead of Amount '2000'
Project '93-34' has overhead of Amount '3000'
Project '93-66' has overhead of Amount '2000'

9. Project has project actual end date Date
Each Project has project actual end date at most one Date
Examples:
Project '93-34' has project actual end date Date '2.3.94'
Project '93-34' has project actual end date Date '1.4.94'
Project '93-66' has project actual end date Date '2.3.94'

10. Project has project duration in months of Number
Each Project has project duration in months of at most one Number
Examples:
Project '93-34' has project duration in months of Number '6'
Project '93-34' has project duration in months of Number '9'
Project '93-66' has project duration in months of Number '6'

11. Project has project leader with Staff
Every Project has project leader with exactly one Staff
Examples:
Project '93-34' has project leader with Staff '2'
Project '93-34' has project leader with Staff '3'
Project '93-66' has project leader with Staff '2'

12. Project has project name Name
Every Project has project name exactly one Name
Examples:
Project '93-34' has project name Name 'Milk Proteins'
Project '93-34' has project name Name 'Restructured Meat'
Project '93-66' has project name Name 'Milk Proteins'

13. Project has proposal discontinue date Date
    Each Project has proposal discontinue date at most one Date
Examples:
    Project '93-34' has proposal discontinue date Date '1.1.94'
    Project '93-34' has proposal discontinue date Date '1.2.94'
    Project '93-66' has proposal discontinue date Date '1.1.94'

14. Project has proposal follow up date Date
    Each Project has proposal follow up date at most one Date
Examples:
    Project '93-34' has proposal follow up date Date '1.1.94'
    Project '93-34' has proposal follow up date Date '1.2.94'
    Project '93-66' has proposal follow up date Date '1.1.94'

15. Project has proposal sent date Date
    Each Project has proposal sent date at most one Date
Examples:
    Project '93-34' has proposal sent date Date '12.12.93'
    Project '93-34' has proposal sent date Date '11.11.93'
    Project '93.66' has proposal sent date Date '12.12.93'

16. Project has start date Date
    Each Project has start date at most one Date
Examples:
    Project '93-34' has start date Date '3.3.94'
    Project '93-34' has start date Date '4.4.94'
    Project '93-66' has start date Date '3.3.94'

17. Project is accepted
Examples:
    Project '93-34' is accepted
    Project '93-66' is accepted

18. Project is undertaken for Client
    Every Project is undertaken for exactly one Client
Examples:
    Project '93-34' is undertaken for Client '54'
    Project '93-34' is undertaken for Client '27'
    Project '93-66' is undertaken for Client '54'

Appendix Five *Revised general ledger (credit) examples*

Objects:

---

1. Account: Nested object-type of the fact "Account_Number contains / is associated with Sub_Account"
2. Amount: Simple object-type identified by 'Amount' and implemented as 'money (15,15)'
3. Client: Simple object-type identified by 'ClientID' and implemented as 'integer'
4. Dept: Simple object-type identified by 'Dept' and implemented as 'smallint'
   range { 16685 }
   description "Default 16685"
5. General_Ledger: Simple object-type identified by 'GeneralLedgerNumber' and implemented as 'smallint'
6. Invoice: Simple object-type identified by 'InvoiceNumber' and implemented as 'integer'
7. Invoice_Date: Simple object-type identified by 'InvoiceDate' and implemented as 'date'
8. Month: Simple object-type identified by 'Month' and implemented as 'smallint'
9. Project: Simple object-type identified by 'ProjectID' and implemented as 'varchar 8'
10. Signal: Simple object-type identified by 'Signal' and implemented as 'varchar 8'
    range { CREDIT .. DEBIT }
11. Year: Simple object-type identified by 'Year' and implemented as 'smallint'

Facts:

---

1. Client is assigned to General_Ledger / General_Ledger receives Client
   Each General_Ledger receives at most one Client

Examples:

Client '52' is assigned to General_Ledger '105'
Client '63' is assigned to General_Ledger '105'
Client '52' is assigned to General_Ledger '110'

2. General_Ledger belongs to Dept
   Every General_Ledger belongs to exactly one Dept

Examples:

General_Ledger '105' belongs to Dept '16685'
General_Ledger '105' belongs to Dept '18820'
General_Ledger '110' belongs to Dept '16685'

3. General_Ledger has credit note Signal
   Every General_Ledger has credit note exactly one Signal

Examples:

General_Ledger '105' has credit note Signal 'CREDIT'
General_Ledger '105' has credit note Signal 'DEBIT'
General_Ledger '110' has credit note Signal 'CREDIT'

4. General_Ledger has invoice amount Amount
   Every General_Ledger has invoice amount exactly one Amount

Examples:

General_Ledger '105' has invoice amount Amount '25000'
General_Ledger '105' has invoice amount Amount '30000'
General_Ledger '110' has invoice amount Amount '25000'

5. General_Ledger is assigned to Account
   Every General_Ledger is assigned to exactly one Account

Examples:

General_Ledger '23' is assigned to Account '02020,FORST'
General_Ledger '23' is assigned to Account '02020,NZDB'
General_Ledger '24' is assigned to Account '02020,FORST'

6. General_Ledger is created in Month
Each General_Ledger is created in at most one Month
Examples:
General_Ledger '105' is created in Month '6'
General_Ledger '105' is created in Month '12'
General_Ledger '110' is created in Month '6'

7. General_Ledger is created in Year
Each General_Ledger is created in at most one Year
Examples:
General_Ledger '105' is created in Year '1994'
General_Ledger '105' is created in Year '1995'
General_Ledger '110' is created in Year '1994'

8. Invoice refers to General_Ledger / General_Ledger has Invoice
Each Invoice refers to at most one General_Ledger and
Each General_Ledger has at most one Invoice
Examples:
Invoice '1056' refers to General_Ledger '105'
Invoice '1056' refers to General_Ledger '110'
Invoice '1164' refers to General_Ledger '105'

9. Invoice_Date is recorded for General_Ledger / General_Ledger is assigned Invoice_Date
Each General_Ledger is assigned at most one Invoice_Date
Examples:
Invoice_Date '21.11.94' is recorded for General_Ledger '105'
Invoice_Date '25.10.94' is recorded for General_Ledger '105'
Invoice_Date '21.11.94' is recorded for General_Ledger '110'

10. Project has General_Ledger / General_Ledger is assigned to Project
Each General_Ledger is assigned to at most one Project
Examples:
Project '93-34' has General_Ledger '105'
Project '93-34' has General_Ledger '110'
Project '93-66' has General_Ledger '105'

Appendix Six *Administration schema diagrams and table structures*

This appendix contains two sets of information about the administration database. The first set are the NIAM type diagrams and resultant table structures for the schema re-design performed by the author. The second set are the NIAM type diagrams and resultant table structures for the conceptual schema design process performed by the associate researcher.

1. NIAM type diagrams for the administration universe of discourse (author's set).

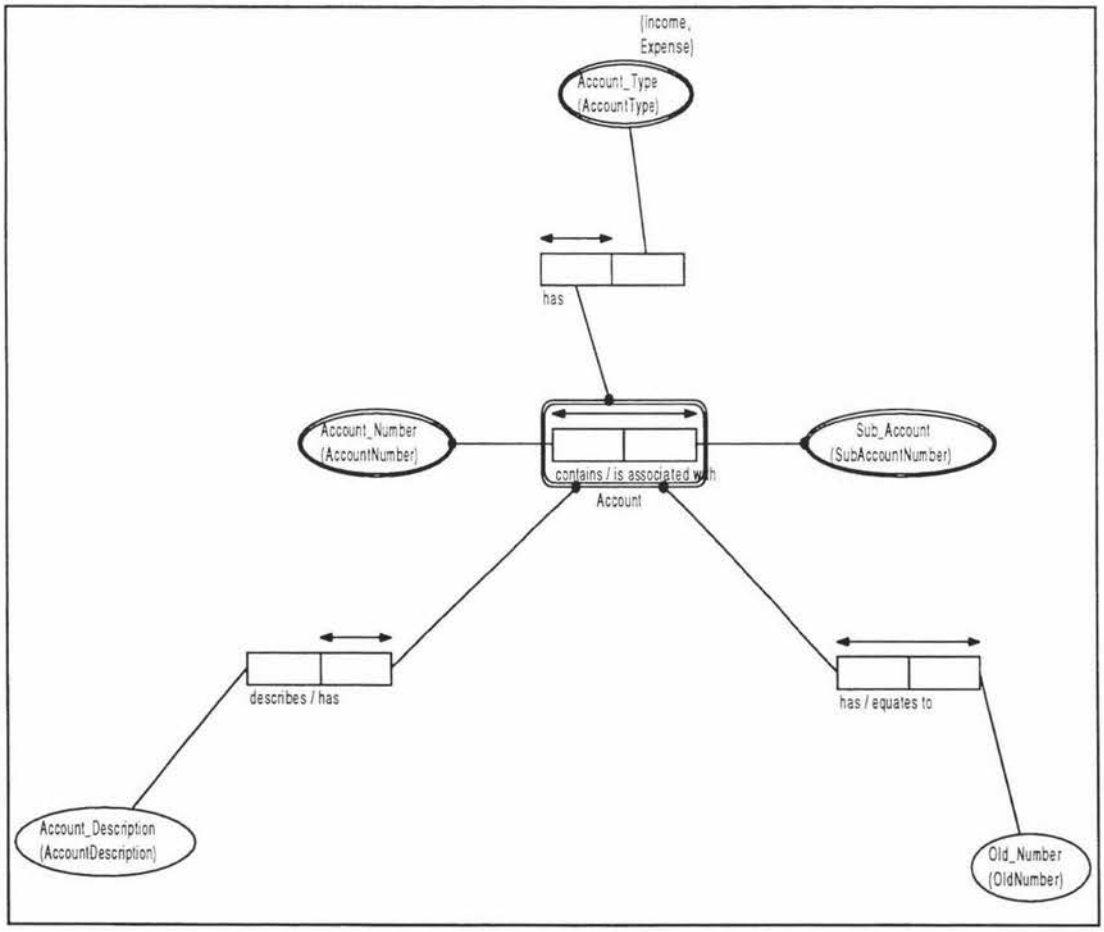


Figure A6.1 Author's Account Information.

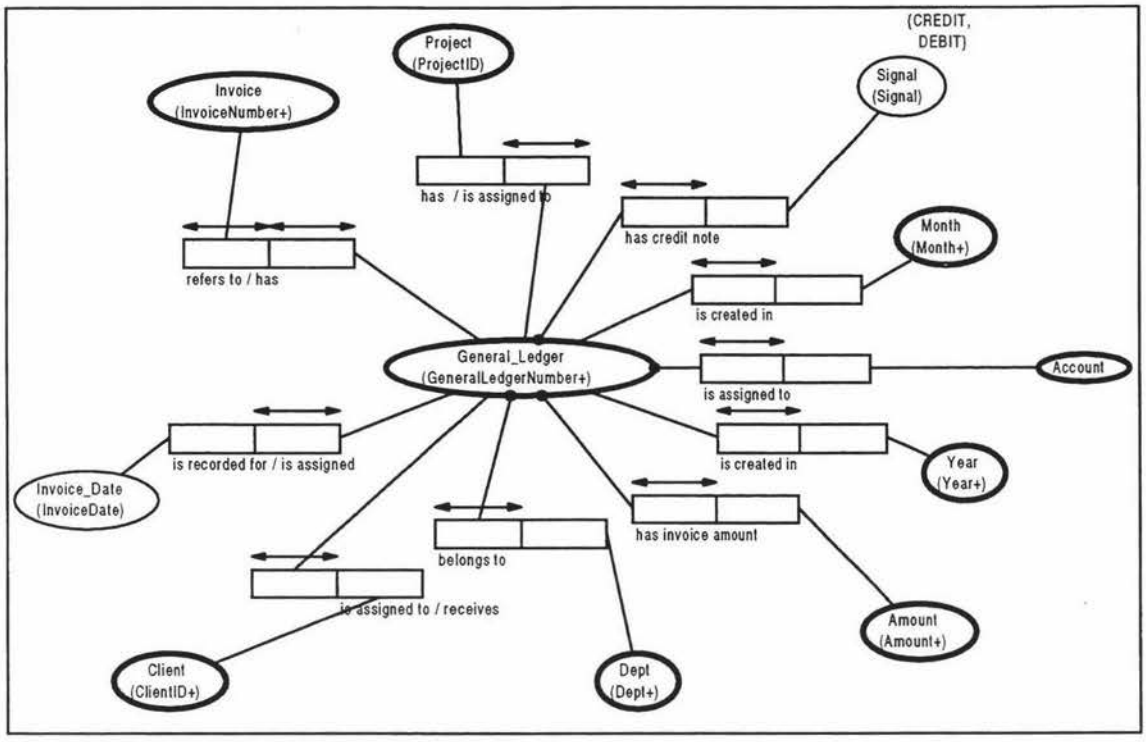


Figure A6.2 Author's Credits Information.

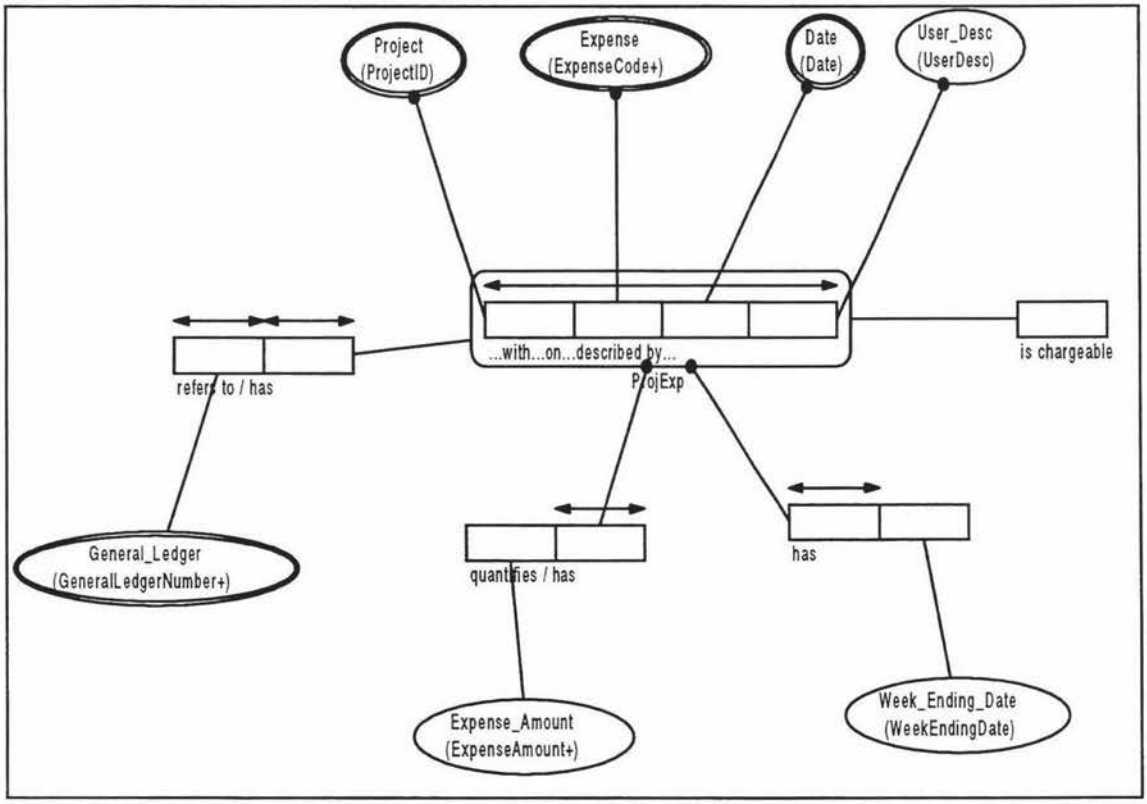


Figure A6.3 Author's Project Expense Information.

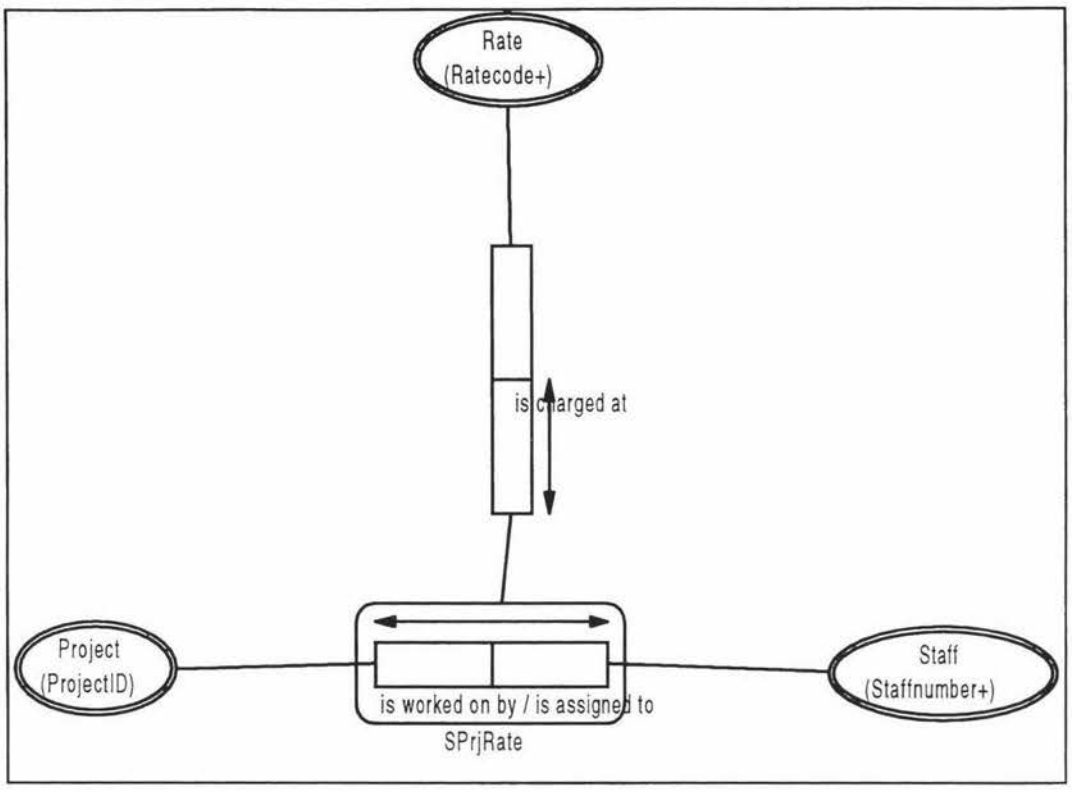


Figure A6.4. Author's Staff, Project and Rate Information.

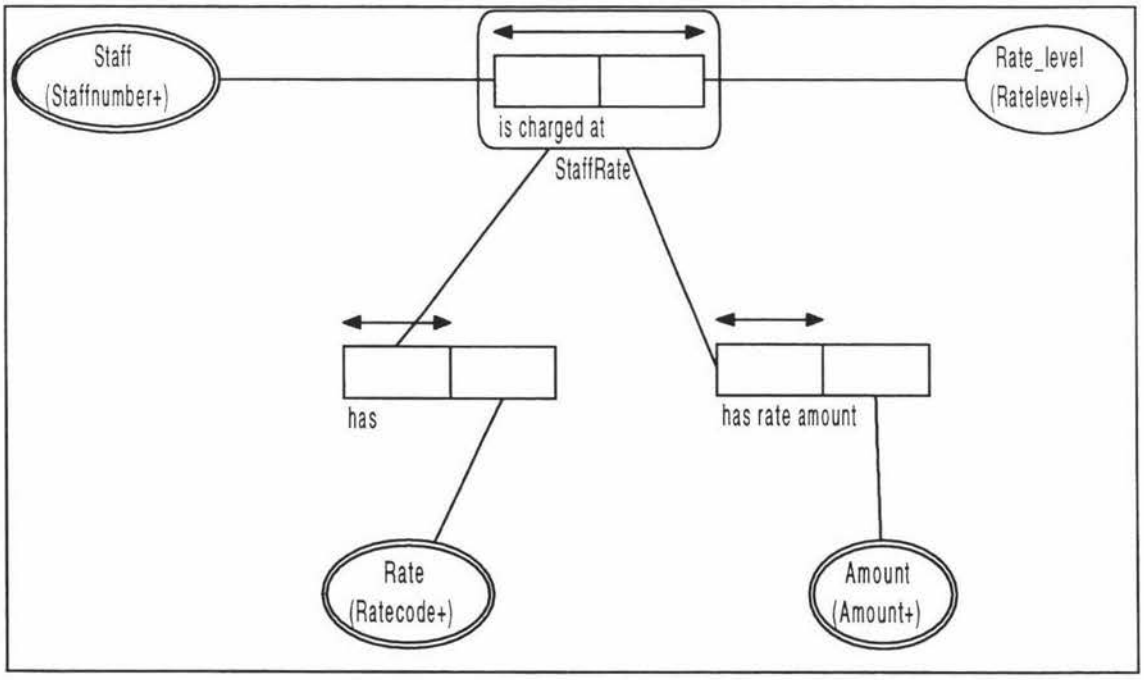


Figure A6.5. Author's Staff Rate Information.

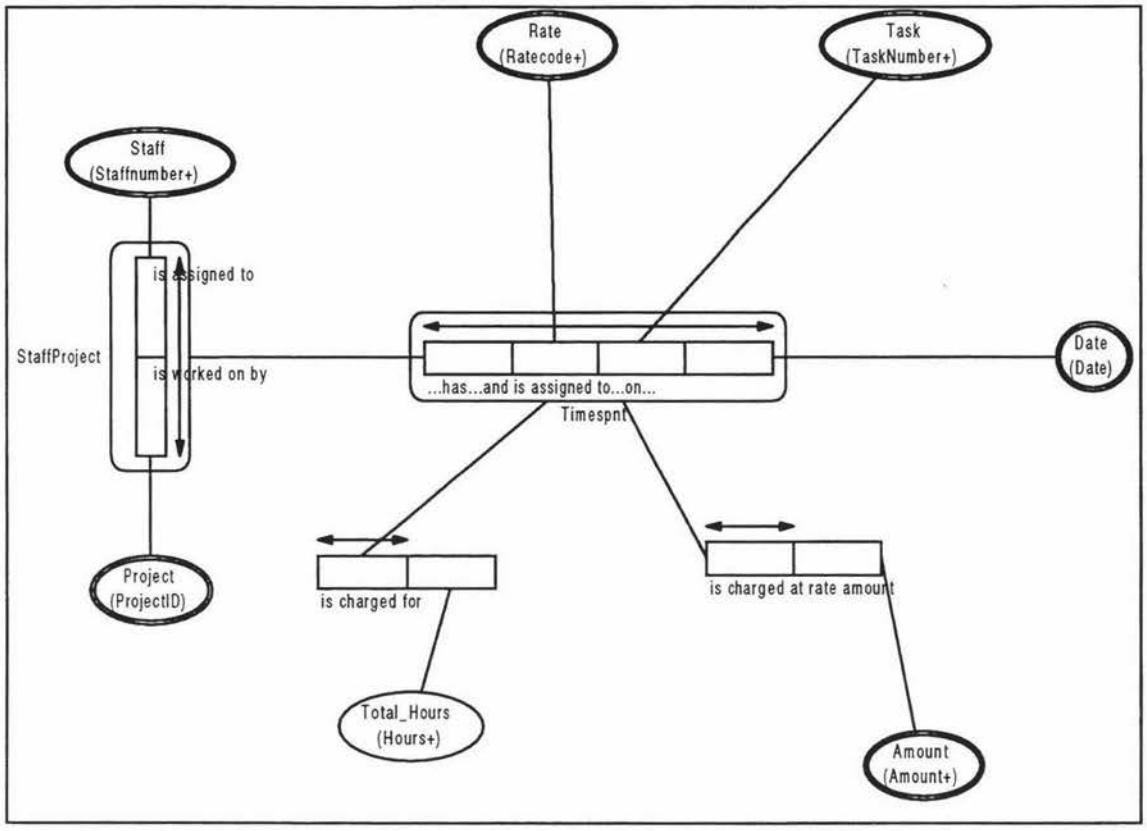


Figure A6.6. Author's Timespent Information.

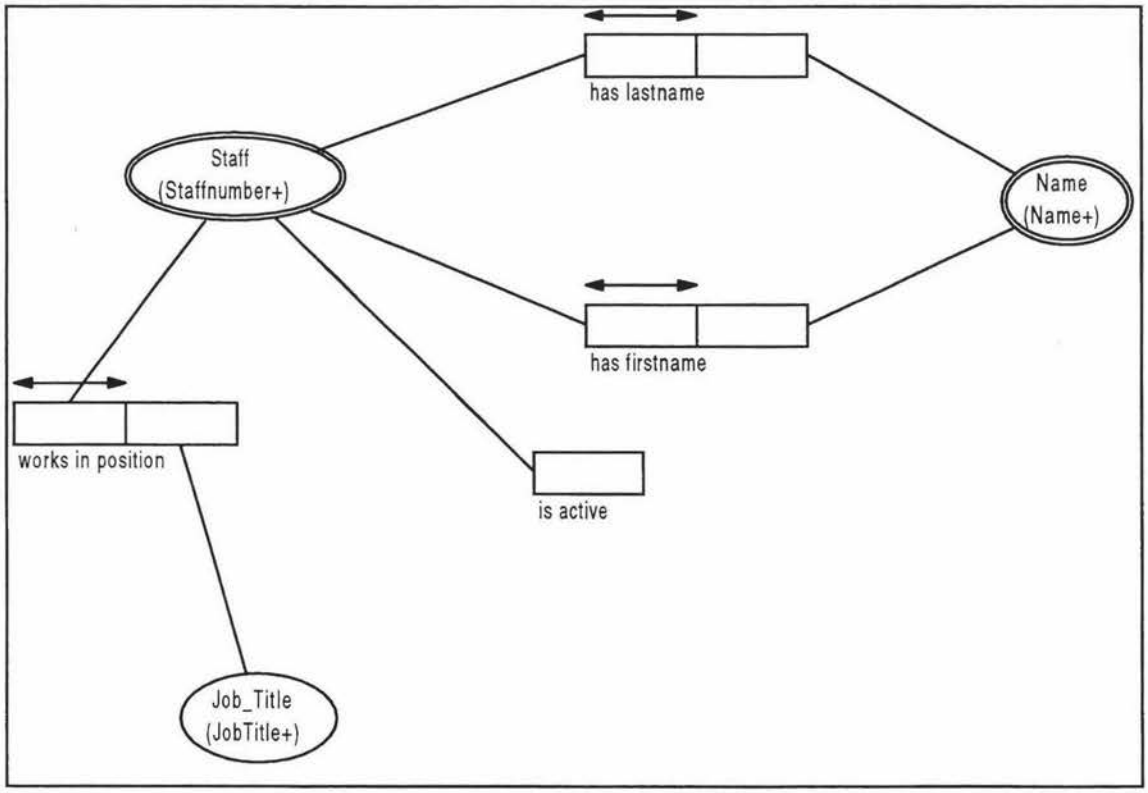


Figure A6.7. Author's Staff Information.

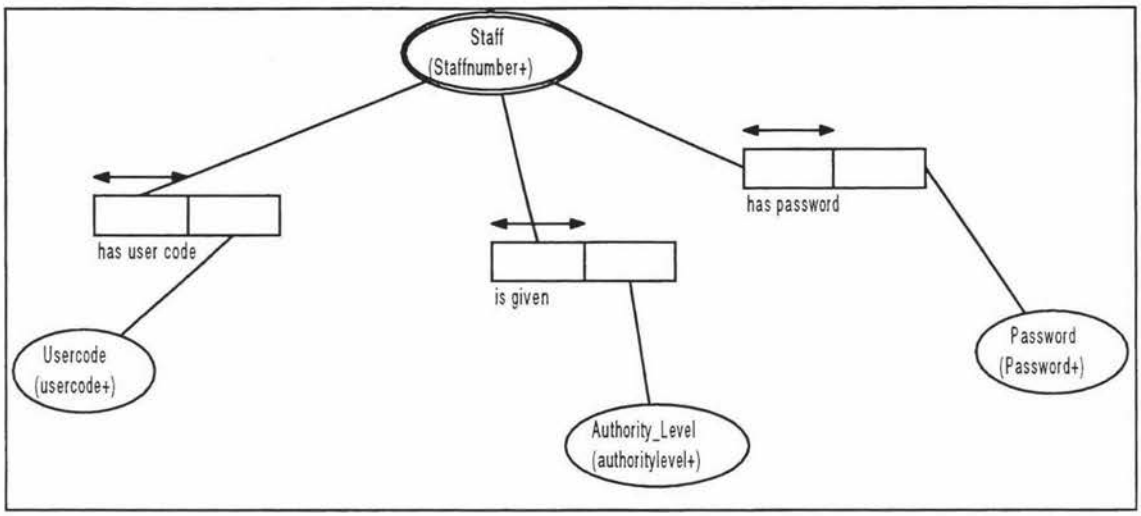


Figure A6.8. Author's Password Information.

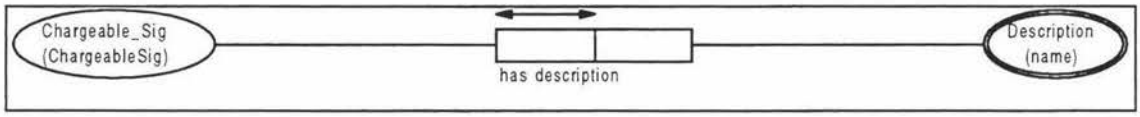


Figure A6.9. Author's Chargeable Codes Information.

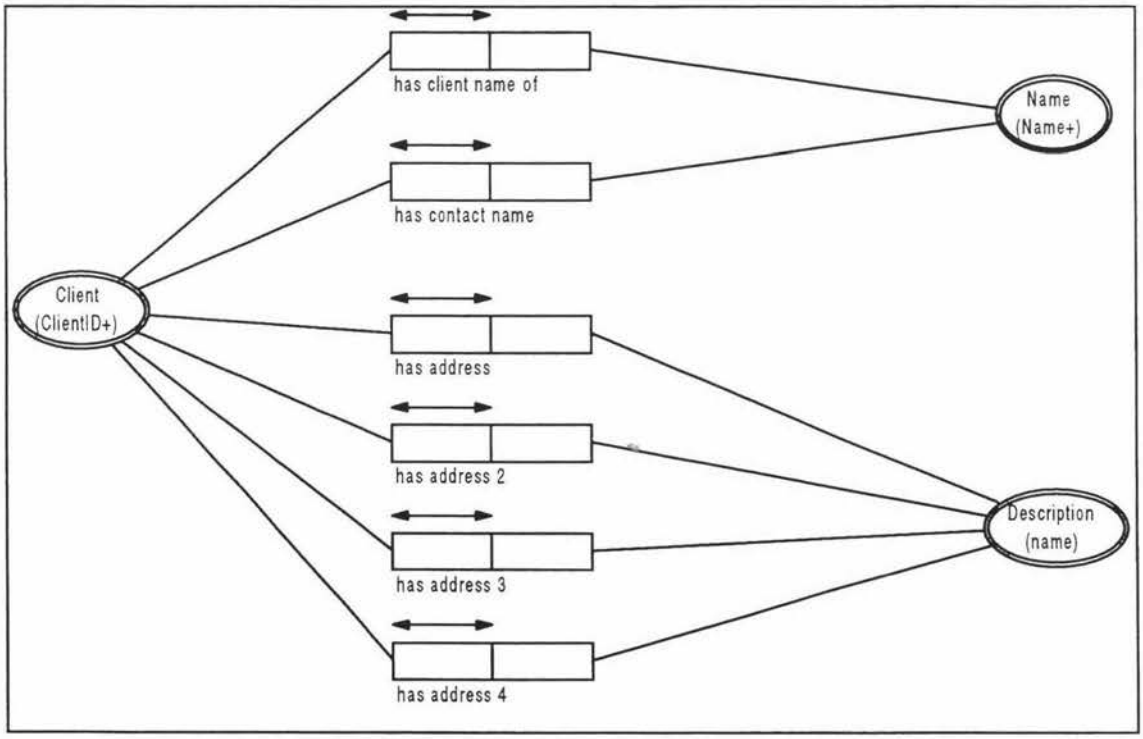


Figure A6.10. Author's Client Information.

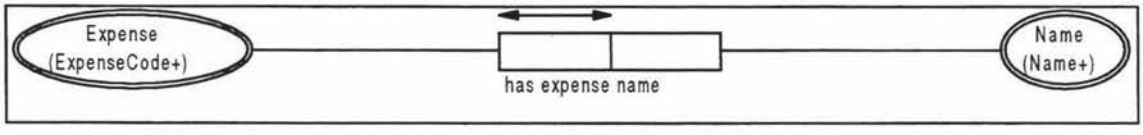


Figure A6.11. Author's Expense Information.

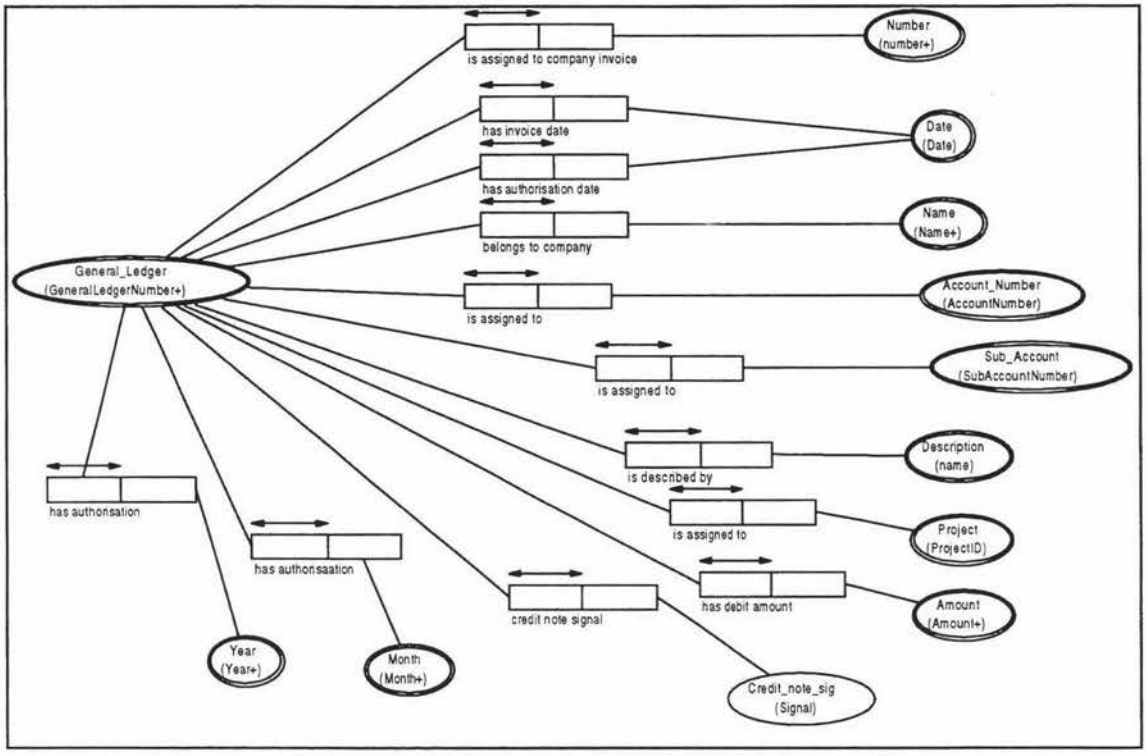


Figure A6.12. Author's Debits Information.

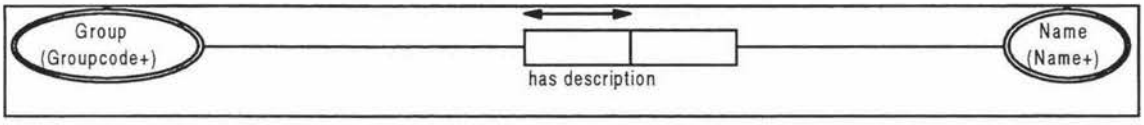


Figure A6.13. Author's Group Code Information.

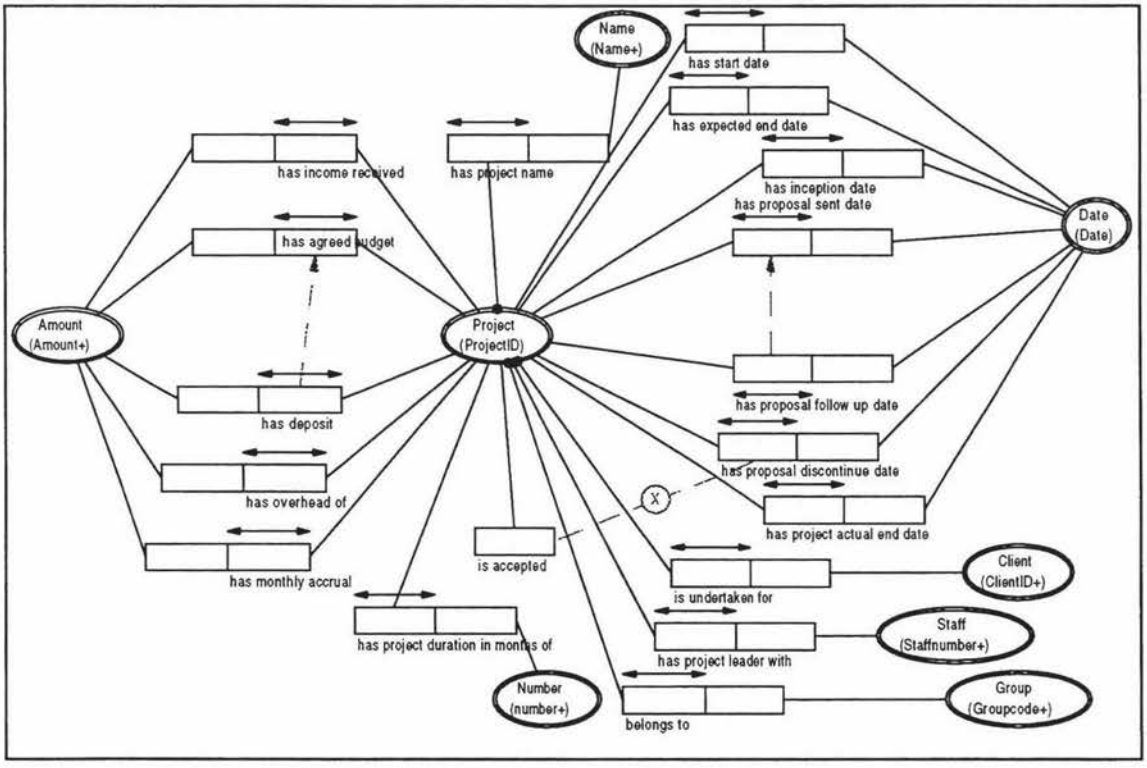


Figure A6.14. Author's Project Information.

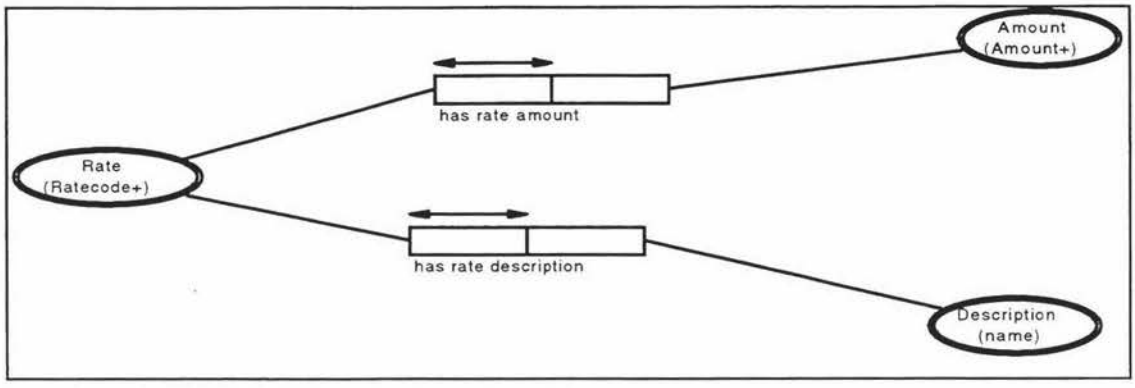


Figure A6.15. Author's Pay Rate Information.

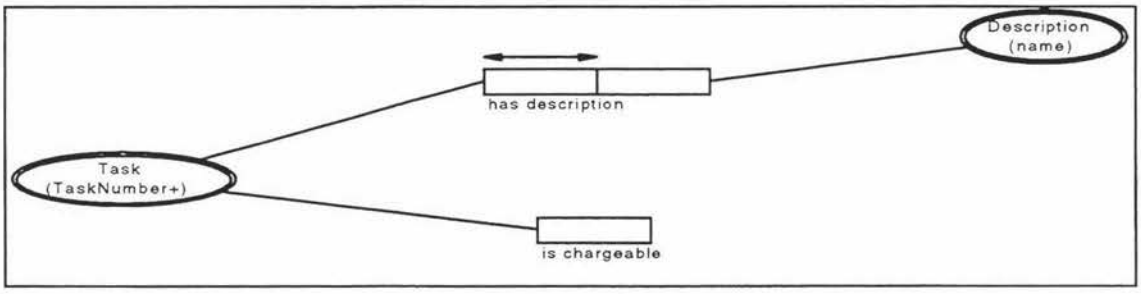


Figure A6.16. Author's Task Information.

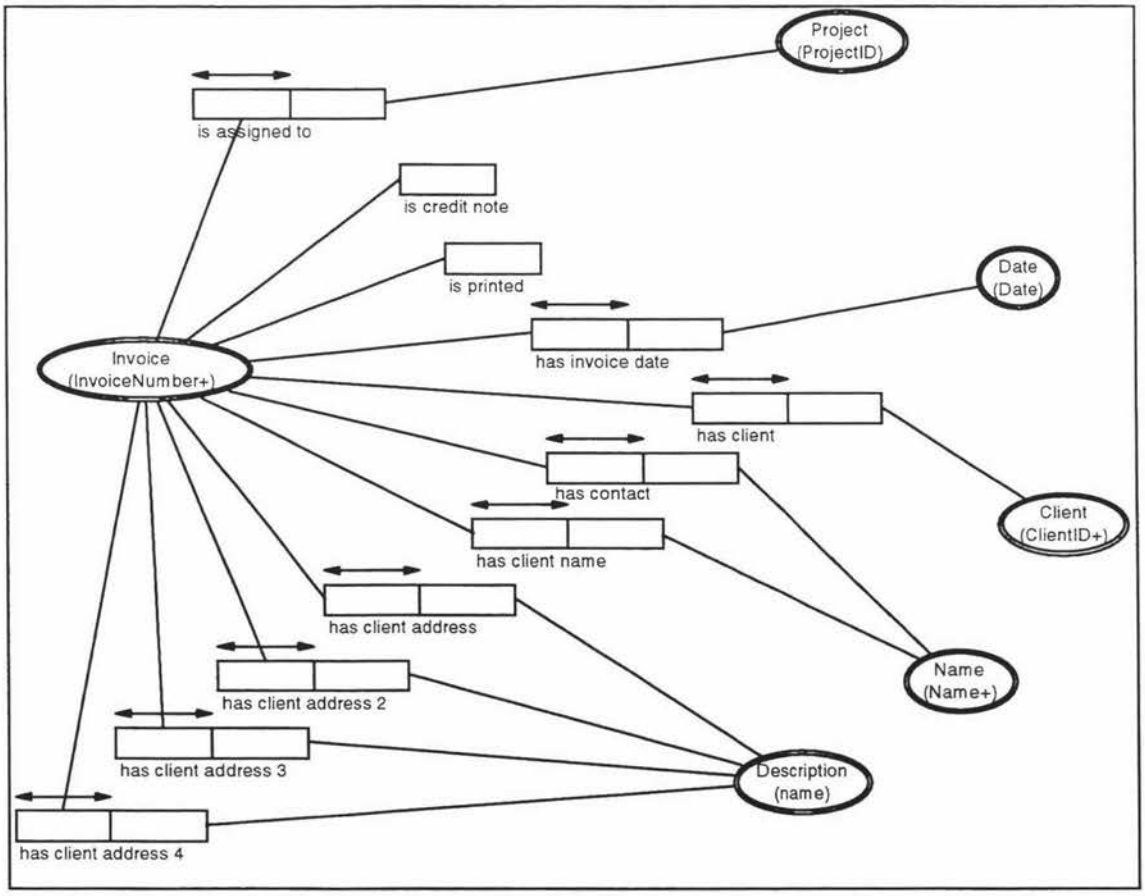


Figure A6.17. Author's Invoice Information (part one).

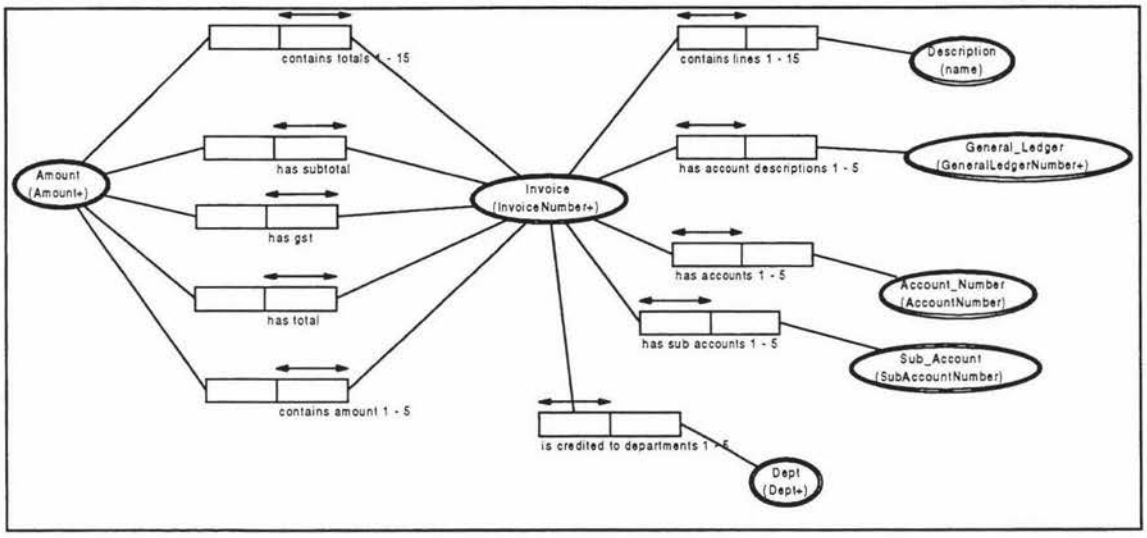


Figure A6.18. Author's Invoice Information (part two).

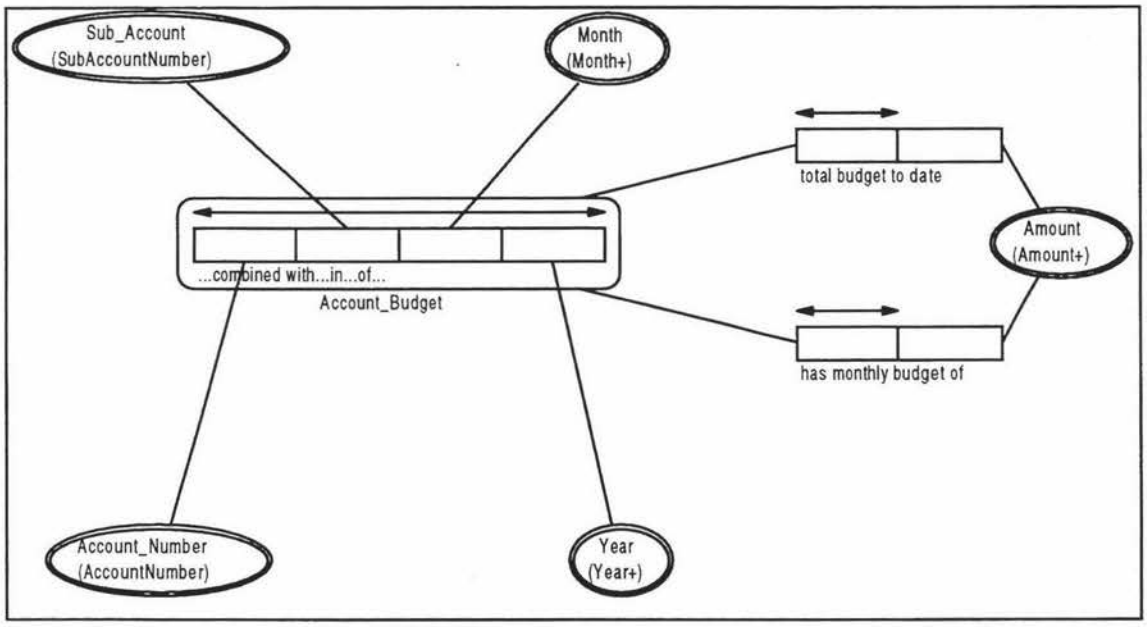


Figure A6.19. Author's Account Budget Information.

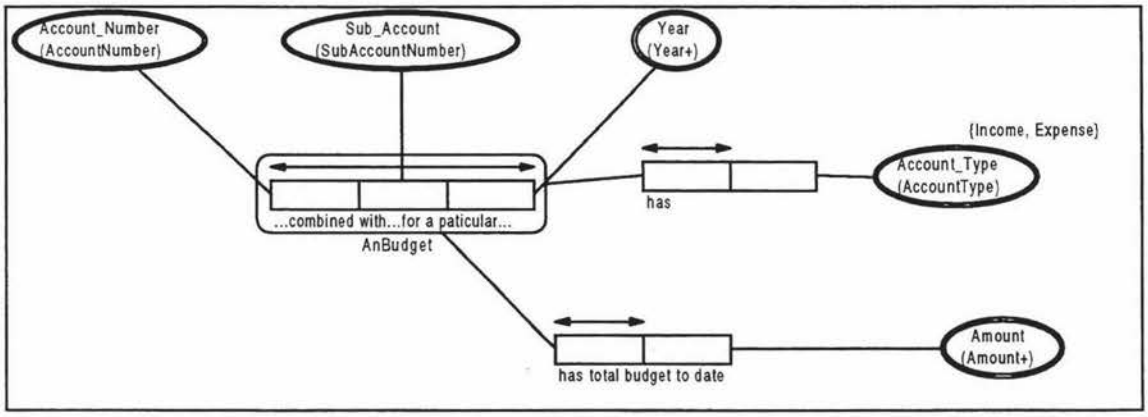


Figure A6.20. Author's Annual Budget Information.

2. Logical schema table structures for the administration universe of discourse (author's set)

Table of Account

Account_Number	varchar (5)	NOT NULL,
Sub_Account	varchar (5)	NOT NULL,
Account_Type	varchar (8)	NOT NULL
Account_Description	varchar (30)	NOT NULL,

PRIMARY KEY (Account_Number, Sub_Account)

Table of Account_Budget

Account_Number	varchar (5)	NOT NULL,
Sub_Account	varchar (5)	NOT NULL,
Month	smallint	NOT NULL,
Year	smallint	NOT NULL,
Monthly_Budget_Amount	money	NULL,
Total_Budget_To_Date	money	NULL,

PRIMARY KEY (Account_Number, Sub_Account, Month, Year)

Account has Old_Number

Account_Number	varchar (5)	NOT NULL,
Sub_Account	varchar (5)	NOT NULL,
Old_Number	varchar (45)	NOT NULL,

PRIMARY KEY (Account_Number, Sub_Account, Old_Number),

Table of AnBudget

Account_Number	varchar (5)	NOT NULL,
Sub_Account	varchar (5)	NOT NULL,
Year	smallint	NOT NULL,
Total_Budget_To_Date	money	NULL,
Account_Type	varchar (8)	NULL

PRIMARY KEY (Account_Number, Sub_Account, Year)

Table of Chargeable_Sig

Chargeable_Sig	varchar (1)	NOT NULL,
Description	string	NULL,

PRIMARY KEY (Chargeable_Sig)

Table of Client

Client	integer	NOT NULL,
Client_Name_Name	integer	NULL,
Contact_Name_Name	integer	NULL,
Address_Description	string	NULL,
Address_2_Description	string	NULL,
Address_3_Description	string	NULL,
Address_4_Description	string	NULL,

PRIMARY KEY (Client)

Table of Expense

Expense code	smallint	NOT NULL,
Expense_Name	integer	NULL,

PRIMARY KEY (Expense code)

Table of General_Ledger

General_Ledger	smallint	NOT NULL,
Account_Number	varchar (5)	NOT NULL,
Sub_Account	varchar (5)	NOT NULL,
Project	varchar (8)	NULL,
Expense	smallint	NULL,
Date	date	NULL
User_Desc	varchar (15)	NULL,
Company_Invoice_Number	integer	NULL,
Invoice_Date	date	NULL
Authorisation_Date	date	NULL
Company_Name	integer	NULL,
Account_Number	varchar (5)	NULL,
Sub_Account	varchar (5)	NULL,
Description	string	NULL,
Debit_Amount	money	NULL,
Authorisation_Month	smallint	NULL,
Authorisation_Year	smallint	NULL,
Credit_Note_Signal	varchar (8)	NOT NULL
Created_Month	smallint	NULL,
Created_Year	smallint	NULL,
Invoice_Amount	money	NOT NULL,
Dept	smallint	NOT NULL
Client	integer	NULL,
PRIMARY KEY (General_Ledger),		

Table of Group

Group	integer	NOT NULL,
Description_Name	integer	NULL,
PRIMARY KEY (Group)		

Table of Invoice

Invoice	integer	NOT NULL,
Project	varchar (8)	NULL,
Credit_Note	logical	NOT NULL,
Printed	logical	NOT NULL,
Invoice_Date	date	NULL
Client_Client	integer	NULL,
Client_Name	integer	NULL,
Client_Address	string	NULL,
Client_Address_2	string	NULL,
Client_Address_3	string	NULL,
Client_Address_4	string	NULL,
Contact_Name	integer	NULL,
Total_1	money	NULL,
.....	money	NULL,
Total_15	money	NULL,
Line_1	string	NULL,
.....	string	NULL,
Line_15	string	NULL,
Subtotal_Amount	money	NULL,
Gst_Amount	money	NULL,
Total_Amount	money	NULL,
General_Ledger_1	smallint	NULL,
.....	smallint	NULL,
General_Ledger_5	smallint	NULL,
Account_Number_1	varchar (5)	NULL,

.....	varchar (5)	NULL,
Account_Number_5	varchar (5)	NULL,
Sub_Account_1	varchar (5)	NULL,
.....	varchar (5)	NULL,
Sub_Account_5	varchar (5)	NULL,
Department_1	smallint	NULL
.....	smallint	NULL
Department_5	smallint	NULL
Amount_1	money	NULL,
.....	money	NULL,
Amount_5	money	NULL,

PRIMARY KEY (Invoice),

Table of ProjExp

Project	varchar (8)	NOT NULL,
Expense	smallint	NOT NULL,
Date	date	NOT NULL
User_Desc	varchar (15)	NOT NULL,
Expense_Amount	money	NOT NULL,
Week_Ending_Date	date	NOT NULL,
Chargeable	logical	NOT NULL,

PRIMARY KEY (Project, Expense, Date, User_Desc),

Table of Project

Project	varchar (8)	NOT NULL,
Project_Name	integer	NOT NULL,
Start_Date	date	NULL
Expected_End_Date	date	NULL
Inception_Date	date	NULL
Proposal_Sent_Date	date	NULL
Proposal_Follow_Date	date	NULL
Proposal_Discontinue_Date	date	NULL
Accepted	logical	NOT NULL,
Project_Actual_End_Date	date	NULL
Agreed_Budget_Amount	money	NULL,
Deposit_Amount	money	NULL,
Overhead_Amount	money	NULL,
Monthly_Accrual_Amount	money	NULL,
Client	integer	NOT NULL,
Income_Received	money	NULL,
Project_Leader	integer	NOT NULL,
Group	integer	NOT NULL,
Project_Duration	integer	NULL,

PRIMARY KEY (Project),

Table of Rate

Rate	integer	NOT NULL,
Rate_Amount	money	NULL,
Rate_Description	string	NULL,

PRIMARY KEY (Rate)

Table of SPrjRate

Project	varchar (8)	NOT NULL,
Staff	integer	NOT NULL,
Rate	integer	NULL,

PRIMARY KEY (Assigned_To_Project, Worked_Staff),

Table of Staff

Staff	integer	NOT NULL,
Firstname	integer	NULL,
Lastname	integer	NULL,
Title	integer	NULL,
Active	logical	NOT NULL,
Password	integer	NULL,
Authority_Level	integer	NULL,
Usercode	integer	NULL,
PRIMARY KEY (Staff)		

Table of StaffRate

Staff	integer	NOT NULL,
Rate	integer	NULL,
Rate_Amount	money	NULL,
PRIMARY KEY (Staff, Charged_Rate_level),		

Table of Task

Task	integer	NOT NULL,
Description	string	NULL,
Chargeable	logical	NOT NULL,
PRIMARY KEY (Task)		

Table of Timespnt

Staff	integer	NOT NULL,
Project	varchar (8)	NOT NULL,
Rate	integer	NOT NULL,
Task	integer	NOT NULL,
Date	date	NOT NULL
Total_Hours	integer	NULL,
Rate_Amount	money	NULL,
PRIMARY KEY (Worked_Staff, Assigned_To_Project, Rate, Task, Date).		

3. NIAM type diagrams for the administration universe of discourse (associate researcher's set).

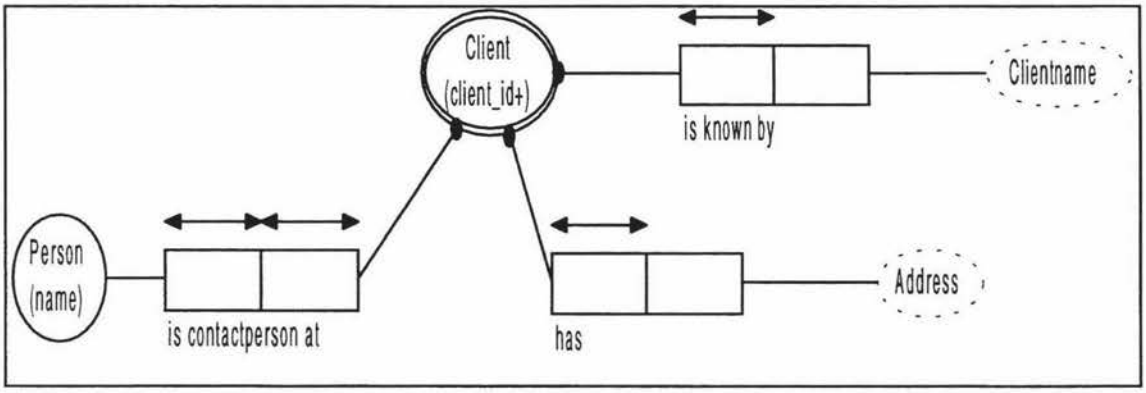


Figure A6.21. Associate Researcher's Client Information.

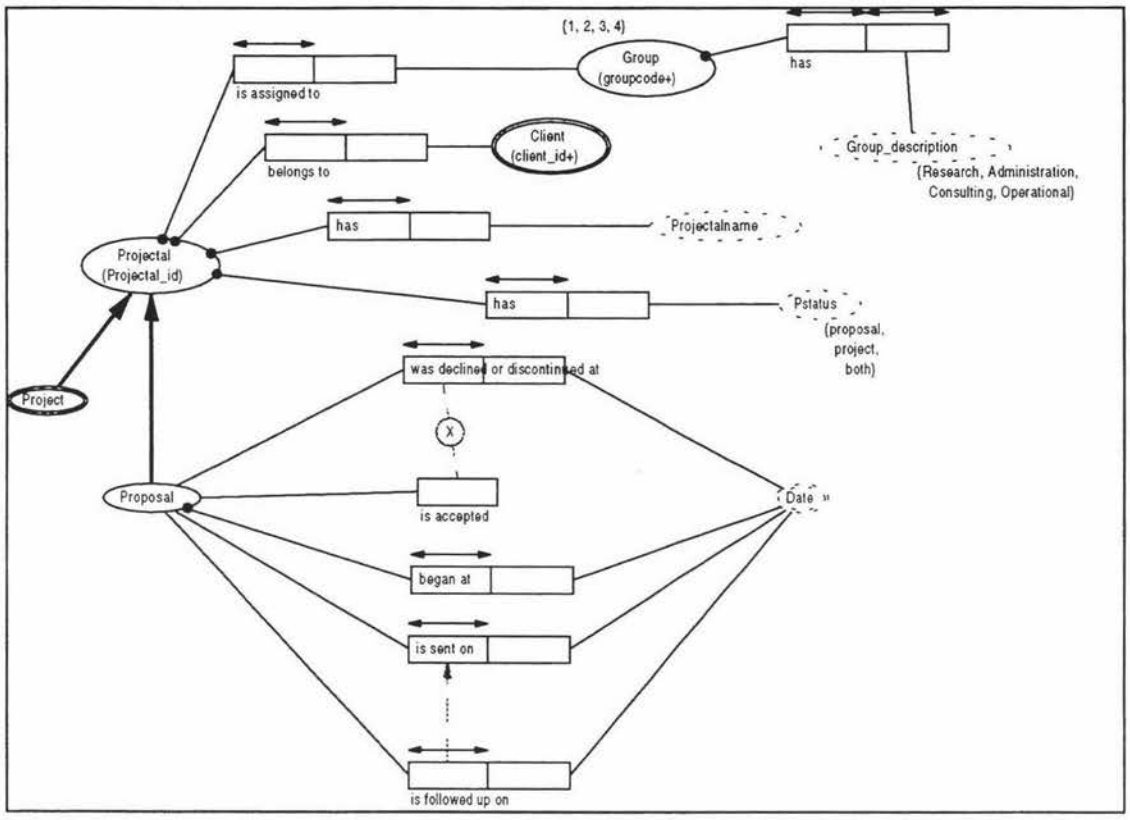


Figure A6.22. Associate Researcher's Proposal Information.

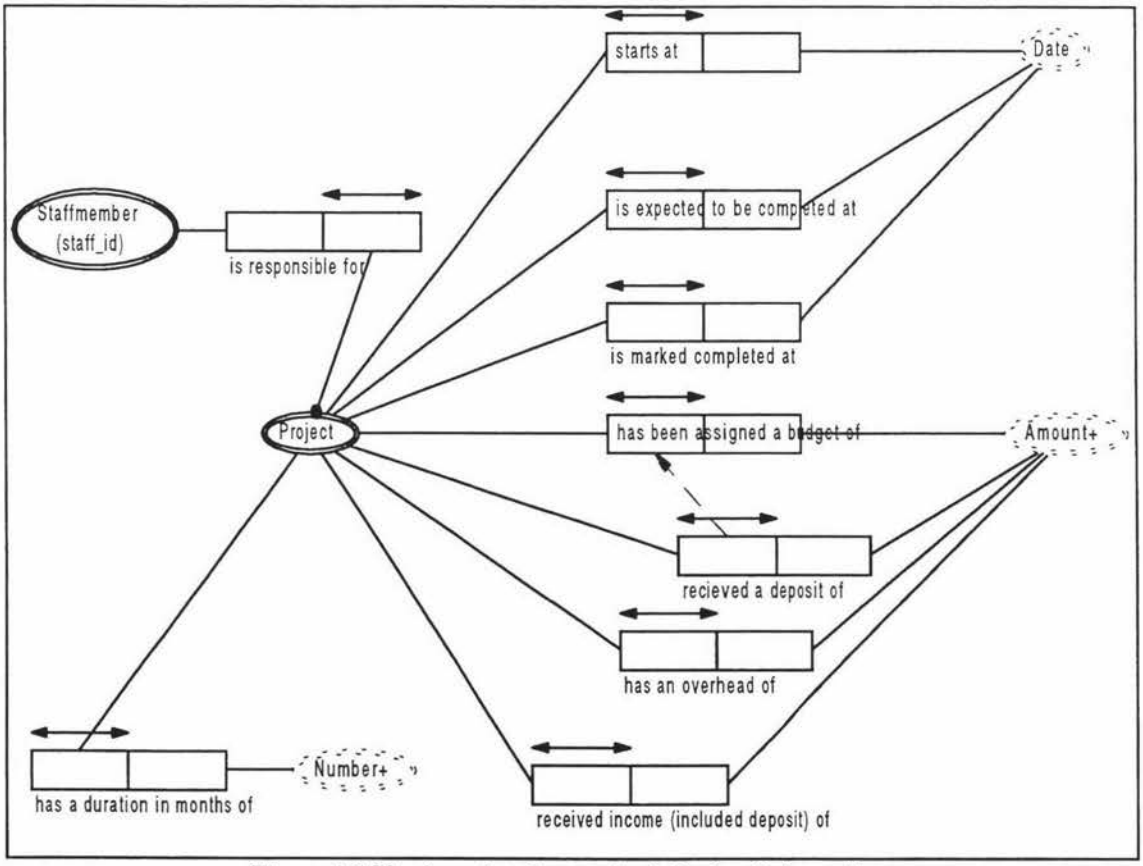


Figure A6.23. Associate Researcher's Project Information.

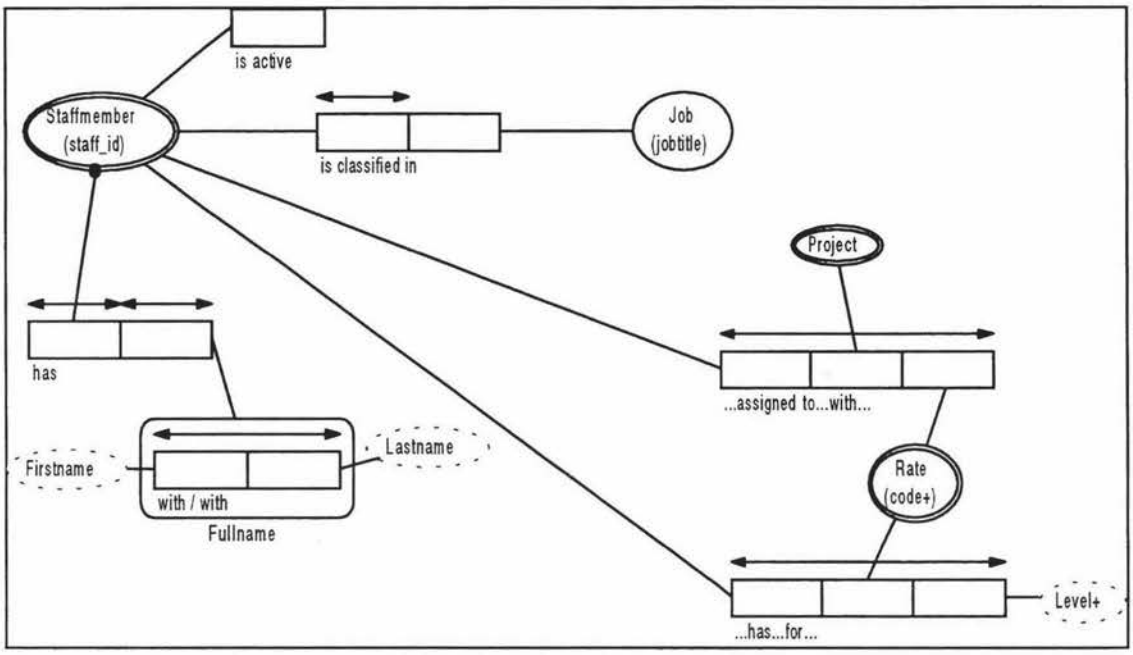


Figure A6.24. Associate Researcher's Staff Information.

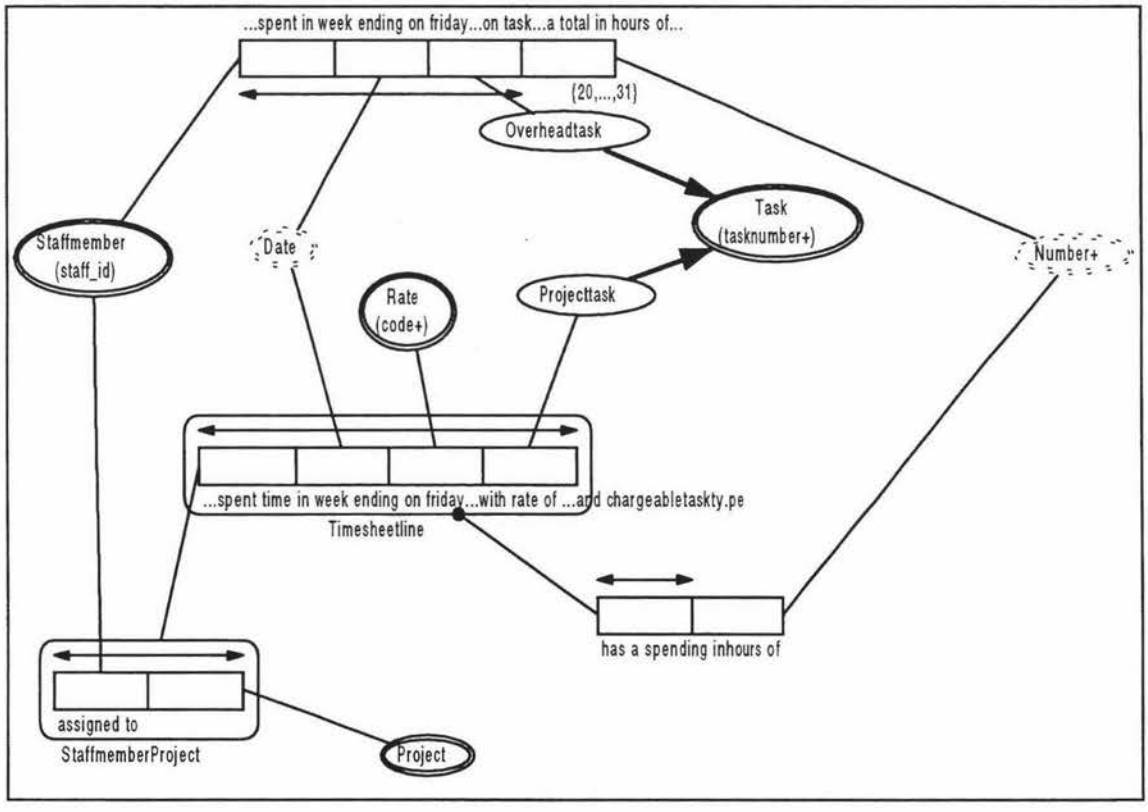


Figure A6.25. Associate Researcher's Timesheet Information.

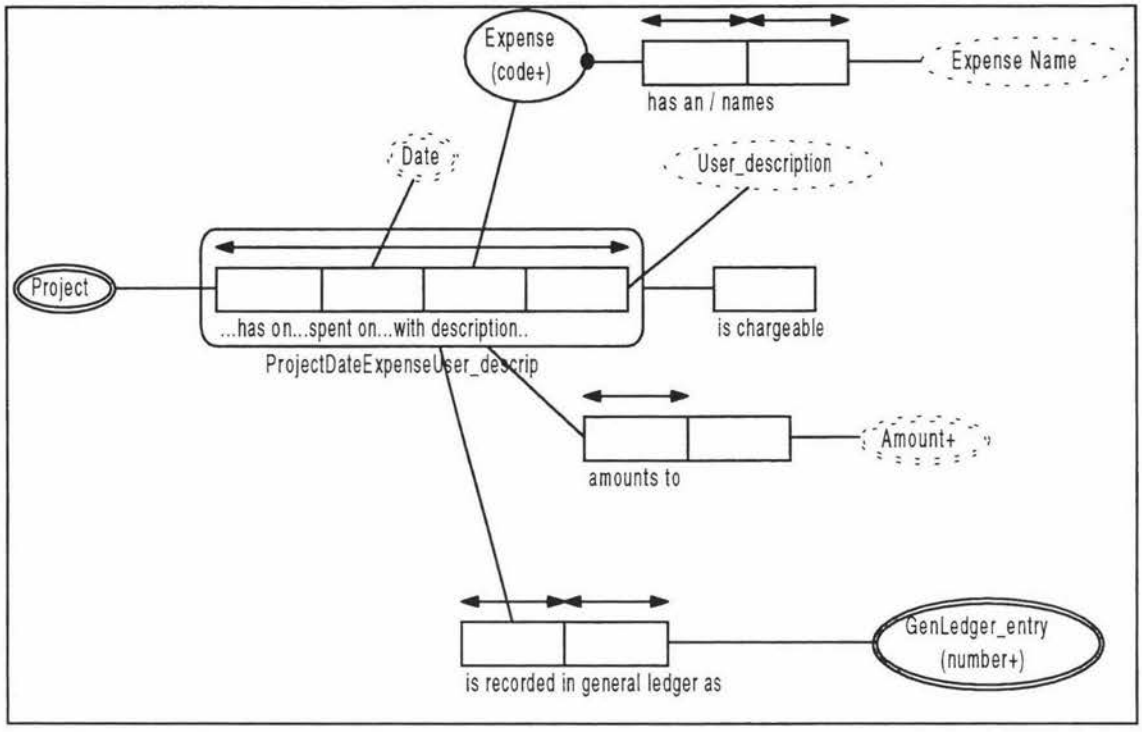


Figure A6.26. Associate Researcher's Expense Information.

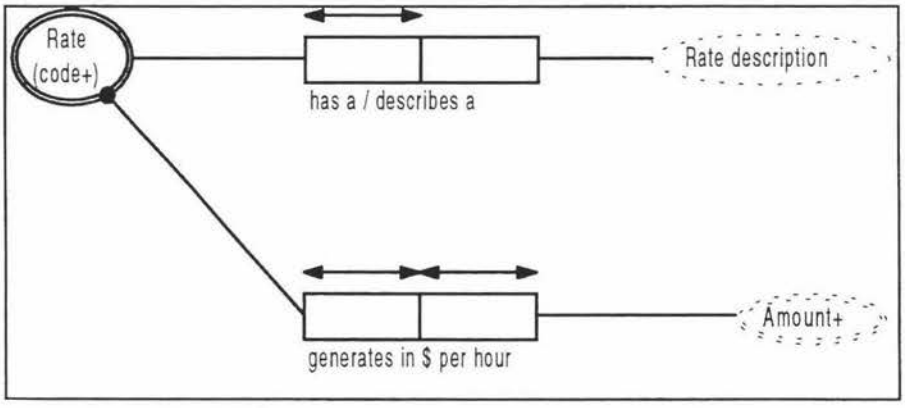


Figure A6.27. Associate Researcher's Rate Information.

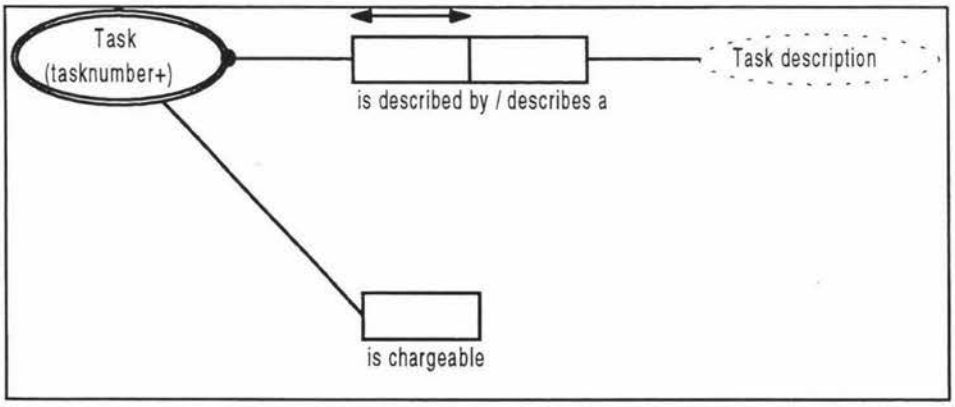


Figure A6.28. Associate Researcher's Task Information.

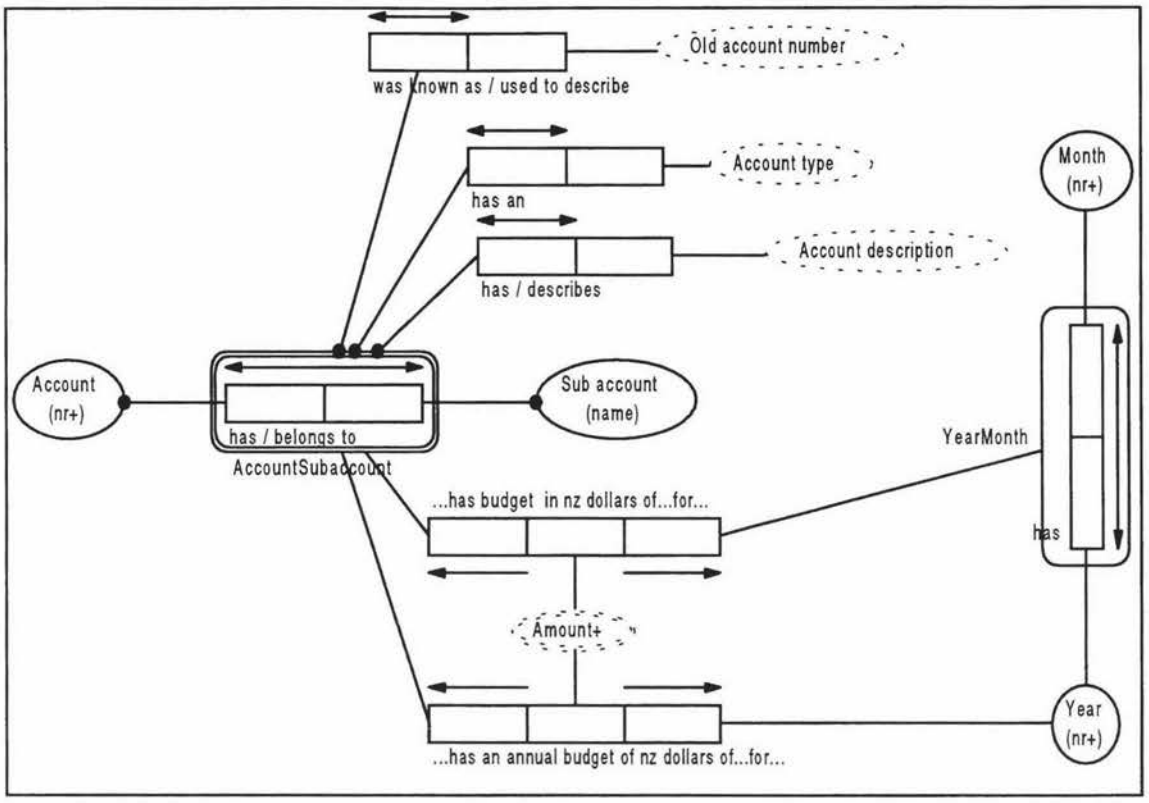


Figure A6.29. Associate Researcher's Account Information.

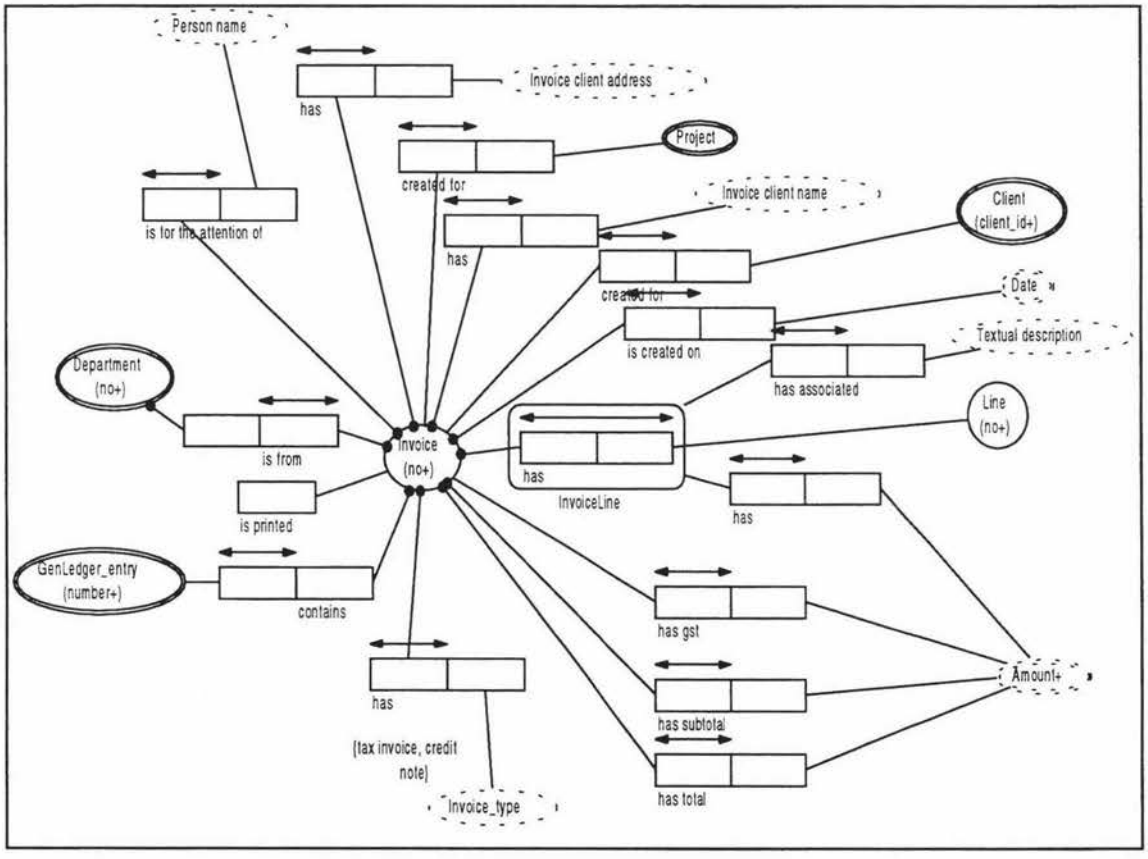


Figure A6.30. Associate Researcher's Invoice Information.

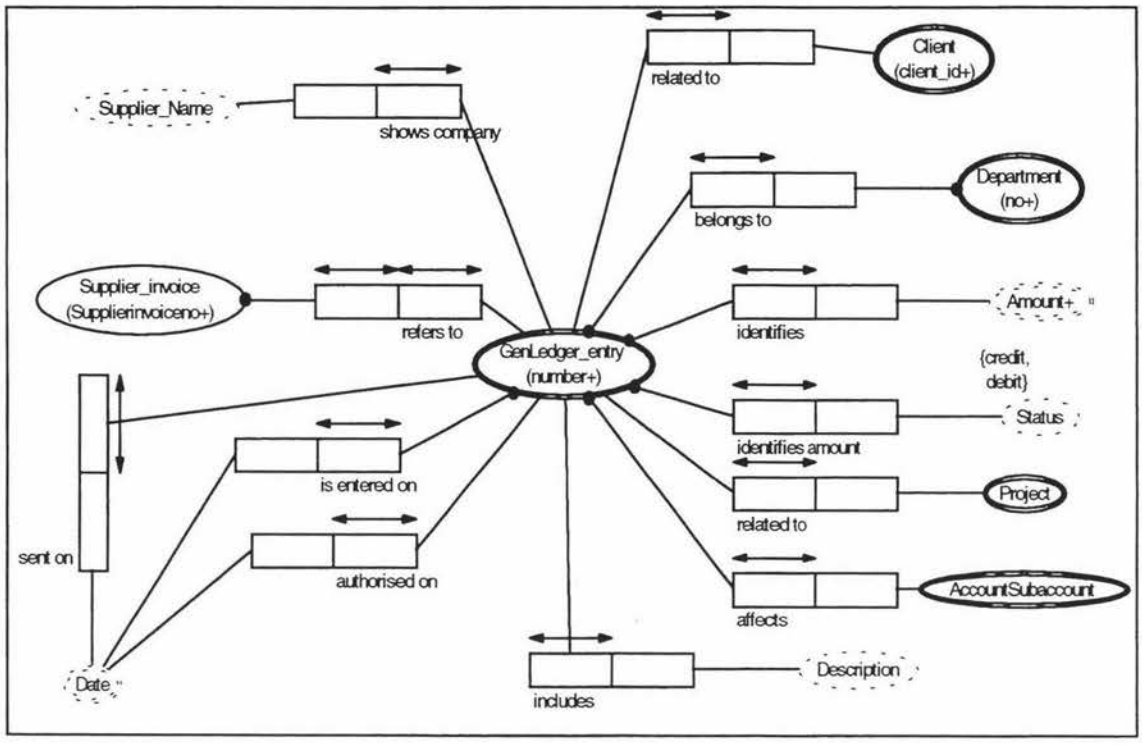


Figure A6.31. Associate Researcher's General Ledger Information

3. Logical schema table structures for the administration universe of discourse (associate researcher's set)

Table of AccountSubaccount

Account	smallint	NOT NULL,
Sub_Account	varchar(20)	NOT NULL,
Account_Description	char varying(20)	NOT NULL,
An_Account_Type	char varying(20)	NOT NULL,
Old_Account_Number	char varying(20)	NOT NULL,

PRIMARY KEY (Account, Sub_Account)

Table of AccountSubaccount_Amnt_Yr

Account	smallint	NOT NULL,
Sub_Account	varchar(20)	NOT NULL,
Amount	money	NOT NULL,
Year	smallint	NOT NULL,

PRIMARY KEY (Account, Sub_Account, Year),

Table of AccountSubacnt_mnt_YrMnth

Account	smallint	NOT NULL,
Sub_Account	varchar(20)	NOT NULL,
Amount	money	NOT NULL,
Year	smallint	NOT NULL,
Month	smallint	NOT NULL,

PRIMARY KEY (Account, Sub_Account, Year, Month),

Table of Client

Client	longinteger	NOT NULL,
Clientname	varchar(25)	NOT NULL,
Address	varchar(50)	NOT NULL,
Person	varchar(20)	NOT NULL,
PRIMARY KEY (Client),		

Table of Expense_Master

Expense	smallint	NOT NULL,
Expense_Name	char varying(20)	NOT NULL,
PRIMARY KEY (Expense),		

Table of GenLedger_entry

GenLedger	longinteger	NOT NULL,
Amount	money	NOT NULL,
Amount_Status	char varying(10)	NOT NULL
Invoice	smallint	NULL,
Department	longint	NOT NULL
Entered_Date	date	NOT NULL,
Project	varchar(4)	NULL,
Account	smallint	NOT NULL,
Sub_Account	varchar(20)	NOT NULL,
Supplier_invoic	integer	NULL,
Sent_Date	date	NULL,
Company_Supplier_Number	char varying(20)	NULL,
Description	char varying(20)	NULL,
Authorised_Date	date	NULL,
Client	longinteger	NULL,
PRIMARY KEY (GenLedger_entry),		

Table of Group_Master

Group	smallint	NOT NULL
Group_description	char(40)	NOT NULL
PRIMARY KEY (Group),		

Table of Invoice

Invoice	smallint	NOT NULL,
Created_Date	date	NOT NULL,
Gst_Amount	money	NULL,
Subtotal_Amount	money	NOT NULL,
Total_Amount	money	NOT NULL,
Department	longint	NOT NULL
Invoice_Client_Name	char varying(20)	NOT NULL,
Invoice_Client_Address	char varying(120)	NOT NULL,
Attention_Person_Name	char varying(20)	NOT NULL,
Invoice_type	char varying(20)	NOT NULL
Project	varchar(4)	NULL,
Printed	logical	NOT NULL,
Client	longinteger	NULL,
PRIMARY KEY (Invoice),		

Table of InvoiceLine

Invoice	smallint	NOT NULL,
Line	smallint	NOT NULL,
Amount	money	NULL,
Description	char varying(40)	NULL,
PRIMARY KEY (Invoice, Line),		

Table of ProjectDateExpenssr_dscrp

Project	varchar(4)	NOT NULL,
Date	date	NOT NULL,
Expense	smallint	NOT NULL,
User_description	char varying(40)	NOT NULL,
Amount	money	NULL,
Chargeable	logical	NOT NULL,
General_Ledger_number	longinteger	NULL,
PRIMARY KEY (Project, Date, Expense, User_description),		

Table of Projectal

Projectal	varchar(4)	NOT NULL,
Group	smallint	NOT NULL
Client	longinteger	NOT NULL,
Projectalname	varchar(50)	NOT NULL,
Pstatus	varchar(20)	NOT NULL
Starts_Date	date	NULL,
Expected_To_Be_Compltd_Date	date	NULL,
Budgt_amount	money	NULL,
Deposit_Amount	money	NULL,
Overhead_Amount	money	NULL,
Duration	integer	NULL,
Completed_Date	date	NULL,
Staffmember	varchar(3)	NULL,
Deposit_amount	money	NULL,
PRIMARY KEY (Projectal),		

Table of Proposal

Proposal	varchar(4)	NOT NULL,
Accepted	logical	NOT NULL,
Declined_Or_Discontind_Date	date	NULL,
Began_Date	date	NULL,
Sent_Date	date	NULL,
Followed_Date	date	NULL,
PRIMARY KEY (Proposal),		

Table of Rate

Rate	smallint	NOT NULL,
Rate_Description	char varying(20)	NULL,
Per_Hour_Amount	money	NOT NULL,
PRIMARY KEY (Rate),		

Table of Staffmember

Staffmember	varchar(3)	NOT NULL,
Classified_Job	varchar(20)	NULL,
Firstname	varchar(20)	NOT NULL,
Lastname	varchar(25)	NOT NULL,
Active	logical	NOT NULL,
PRIMARY KEY (Staffmember),		

Table of Staffmember_Project_Rate

Staffmember	varchar(3)	NOT NULL,
Project	varchar(4)	NOT NULL,
Rate	smallint	NOT NULL,
PRIMARY KEY (Staffmember, Project, Rate),		

Table of Staffmember_Rate_Level

Staffmember	varchar(3)	NOT NULL,
Rate	smallint	NOT NULL,
Level	smallint	NOT NULL,
PRIMARY KEY (Staffmember, Rate, Level),		

Table of Staff_overhead

Staffmember	varchar(3)	NOT NULL,
Date	date	NOT NULL,
Overheadtask	smallint	NOT NULL
Number	integer	NOT NULL,
PRIMARY KEY (Staffmember, Date, Overheadtask),		

Table of Task

Task	smallint	NOT NULL,
Described_Task_Descriptin	char varying(20)	NOT NULL,
Chargeable	logical	NOT NULL,
PRIMARY KEY (Task)		

Table of Timesheetline

Staffmember	varchar(3)	NOT NULL,
Project	varchar(4)	NOT NULL,
Date	date	NOT NULL,
Rate	smallint	NOT NULL,
Projecttask	smallint	NOT NULL,
Hours	integer	NOT NULL,
PRIMARY KEY (Staffmember, Project, Date, Rate, Projecttask),		

Appendix Seven *Example Sentence Sets with Verification*

Objects:

---

1. Address_1: Simple object-type identified by 'desc' and implemented as 'varchar 50'
2. Address_2: Simple object-type identified by 'desc' and implemented as 'varchar 50'
3. Bar_Code: Simple object-type identified by 'BarCode' and implemented as 'blob'
4. Comment: Simple object-type identified by 'Comment' and implemented as 'varchar 255'
5. Date: Simple object-type identified by 'Date' and implemented as 'date'
6. Description: Simple object-type identified by 'Description' and implemented as 'varchar 50'
7. Equip: Simple object-type identified by 'EquipID' and implemented as 'varchar 50'
8. Equipment: Simple object-type identified by 'Equipment' and implemented as 'varchar 50'
9. Fax: Simple object-type identified by 'Fax' and implemented as 'text'
10. History: Simple object-type identified by 'History' and implemented as 'text'
11. Location: Simple object-type identified by 'LocationId' and implemented as 'varchar 50'
12. Manager: Simple object-type identified by 'Manager' and implemented as 'varchar 50'
13. Manufacturer: Simple object-type identified by 'ManufacturerID' and implemented as 'integer'
14. Manufacturers_Name: Simple object-type identified by 'ManufacturersName' and implemented as 'varchar 50'
15. Manufacturers_Serial: Simple object-type identified by 'SerialNumber' and implemented as 'integer'
16. Model: Simple object-type identified by 'Model' and implemented as 'varchar 50'
17. Module: Simple object-type identified by 'ModuleID' and implemented as 'integer'
18. Name: Simple object-type identified by 'Name' and implemented as 'varchar 20'
19. Owner: Simple object-type identified by 'Owner' and implemented as 'varchar 50'
20. Part_Number: Simple object-type identified by 'PartNumber' and implemented as 'varchar 50'
21. Password: Simple object-type identified by 'Password' and implemented as 'varchar 20'
22. Past_Locations: Simple object-type identified by 'PastLocations' and implemented as 'varchar 255'
23. Person: Simple object-type identified by 'PersonId' and implemented as 'varchar 3'
24. Phone: Simple object-type identified by 'Phone' and implemented as 'text'
25. Repair_Agent: Simple object-type identified by 'RepairAgent' and implemented as 'integer'
26. Serial: Simple object-type identified by 'Serial' and implemented as 'integer'
27. Serial_Number: Simple object-type identified by 'serialnumber' and implemented as 'varchar 50'
28. Serial_NumberLocationStatusDat: Nested object-type
29. Site: Simple object-type identified by 'SiteId' and implemented as 'varchar 50'
30. Site_Name: Simple object-type identified by 'SiteName' and implemented as 'varchar 50'
31. Status: Simple object-type identified by 'Status' and implemented as 'varchar 50'
32. Team: Simple object-type identified by 'TeamID' and implemented as 'varchar 3'
33. Teams_name: Simple object-type identified by 'TeamName' and implemented as 'varchar 50'
34. Town: Simple object-type identified by 'Town' and implemented as 'text'
35. Type: Simple object-type identified by 'Desc' and implemented as 'varchar 50'
36. Type_Number: Simple object-type identified by 'TypeNumber' and implemented as 'integer'
37. Usage: Simple object-type identified by 'UsageID' and implemented as 'integer'
38. Usage_description: Simple object-type identified by 'UsageDescription' and implemented as 'varchar 50'
39. Use_Manufacture: Simple object-type identified by 'UseManufacture' and implemented as 'integer'

Facts:

(Sentences in bold italic are verified as false by the universe of discourse expert. Additional comments are also in bold italic font).

1. Equip has Description

Each Equip has at most one Description

Examples:

Equip 'CCA15A' has Description 'CCA 15 BSBD Switch'

Equip 'CCA15A' has Description 'Ch1 Ladder'
Equip 'CCA15B' has Description 'CCA 15 BSBD Switch'

2. Equip is used for Usage
Each Equip is used for at most one Usage
Examples:
Equip 'CCA15A' is used for Usage '0'
Equip 'CCA15A' is used for Usage '2'
Equip 'CCA15B' is used for Usage '0'

3. Equipment is located at Location
Each Location has at most one Equipment that is located at it
Examples:
Equipment 'Test04' is located at Location 'ABF Test04'
Equipment 'Test04' is located at Location 'ABF Test06'
Equipment 'Test06' is located at Location 'ABF Test04'

4. Location contains Equip
Each Location contains at most one Equip
Examples:
Location 'ABF Test04' contains Equip 'Test04'
Location 'ABF Test04' contains Equip 'Test06'
Location 'ABF Test06' contains Equip 'Test04'

5. Location has print label
Examples:
Location 'ABF Test04' has print label
Location 'ABF Test06' has print label

6. Location is contained in Site
Each Location is contained in at most one Site
Examples:
Location 'ABF Test04' is contained in Site 'ABF'
Location 'ABF Test04' is contained in Site 'ACT'
Location 'ABF Test06' is contained in Site 'ABF'

7. Location is used for Usage
Each Location is used for at most one Usage
Examples:
Location 'ABF Test04' is used for Usage '02'
Location 'ABF Test04' is used for Usage '05'
Location 'ABF Test06' is used for Usage '02'

8. Manufacturer is named as Manufacturers_Name
Each Manufacturer is named as at most one Manufacturers_Name
Examples:
Manufacturer '000' is named as Manufacturers_Name 'NEC'
Manufacturer '000' is named as Manufacturers_Name 'Marconi'
Manufacturer '001' is named as Manufacturers_Name 'NEC'

9. Module belongs to Model
Each Module belongs to at most one Model
Examples:
Module '000030000' belongs to Model '36 Mux System'
Module '000030000' belongs to Model '71F2'
Module '222222222' belongs to Model '36 Mux System'

10. Module has type designation Type
    Each Module has type designation at most one Type
Examples:
    Module '000030000' has type designation Type 'Mux'
    Module '000030000' has type designation Type 'Limiter Discriminator'
    Module '222222222' has type designation Type 'Mux'

11. Module has type Type_Number
    Each Module has type at most one Type_Number
Examples:
    Module '000030000' has type Type_Number '000'
    Module '000030000' has type Type_Number '001'
    Module '222222222' has type Type_Number '000'

12. Module has Use_Manufacture
    Each Module has at most one Use_Manufacture
Examples:
    Module '000030000' has Use_Manufacture '00003'
    Module '000030000' has Use_Manufacture '22333'
    Module '222222222' has Use_Manufacture '00003'

13. Module is assigned Part_Number
    Each Module is assigned at most one Part_Number
Examples:
    Module '000030000' is assigned Part_Number '3612A'
    Module '000030000' is assigned Part_Number '80798-M1'
    Module '222222222' is assigned Part_Number '3612A'

14. Module is made by Manufacturer
    Each Module is made by at most one Manufacturer
Examples:
    Module '000030000' is made by Manufacturer '003'
    Module '000030000' is made by Manufacturer '444'
    Module '222222222' is made by Manufacturer '003'

15. Module is repaired by Repair_Agent
    Each Module is repaired by at most one Repair_Agent
Examples:
    Module '000030000' is repaired by Repair_Agent 'BCL Network Support'
    Module '000030000' is repaired by Repair_Agent 'BCL District Teams'
    Module '222222222' is repaired by Repair_Agent 'BCL Network Support'

16. Module is used for Usage
    Each Module is used for at most one Usage
Examples:
    Module '000030000' is used for Usage '00'
    Module '000030000' is used for Usage '44'
    Module '222222222' is used for Usage '00'

17. Owner has fax number Fax
    Each Owner has fax number at most one Fax
Examples:
    Owner 'BCL' has fax number Fax '0-4-3825123'
    Owner 'BCL' has fax number Fax '0-9-2358974'
    Owner 'TVNZ' has fax number Fax '0-4-3825123'

18. Owner has phone number Phone
Each Owner has phone number at most one Phone
Examples:
Owner 'BCL' has phone number Phone '0-4-3825000'
Owner 'BCL' has phone number Phone '0-9-3216544'
Owner 'TVNZ' has phone number Phone '0-4-3825000'

19. Owner is located at Address_1
Each Owner is located at at most one Address_1
Examples:
Owner 'BCl' is located at Address_1 'BCL House'
Owner 'BCL' is located at Address_1 'Symonds Street'
Owner 'TVNZ' is located at Address_1 'BCL House'

20. Owner is located at Address_2
Each Owner is located at at most one Address_2
Examples:
Owner 'BCL' is located at Address_2 '147 Troy Street'
Owner 'BCL' is located at Address_2 '2 Massey Street'
Owner 'TVNZ' is located at Address_2 '147 Troy Street'

21. Owner resides in Town
Each Owner resides in at most one Town
Examples:
Owner 'BCL' resides in Town 'Wellington'
Owner 'BCL' resides in Town 'Auckland'
Owner 'TVNZ' resides in Town 'Wellington'

22. Person known as Name belongs to Team has Password
Each Person, Name, Team, Password combination is unique
Examples:
Person 'MR' known as Name 'Mike Ryder' belongs to Team 'MUB' has Password 'Mike'
Person 'MR' known as Name 'Mike Ryder' belongs to Team 'MUB' has Password 'Smith'
Person 'MR' known as Name 'Mike Ryder' belongs to Team 'WTE' has Password 'Mike'
Person 'MR' known as Name 'John Smith' belongs to Team 'MUB' has Password 'Mike'
Person 'JS' known as Name 'Mike Ryder' belongs to Team 'MUB' has Password 'Mike'

23. Repair_Agent can be phoned on Phone
Each Repair_Agent can be phoned on at most one Phone
Examples:
Repair_Agent 'BCL District Teams' can be phoned on Phone '0-4-1234567'
Repair_Agent 'BCL District Teams' can be phoned on Phone '0-9-4567966'
Repair_Agent 'Clear Communications' can be phoned on Phone '0-4-1234567'

24. Repair_Agent has address line Address_1
Each Repair_Agent has address line at most one Address_1
Examples:
Repair_Agent 'BCL District Teams' has address line Address_1 'BCL House'
Repair_Agent 'Clear Communications' has address line Address_1 'BCL House'
Repair_Agent 'BCL District Teams' has address line Address_1 '49 Symonds Street'

25. Repair_Agent has address line Address_2
Each Repair_Agent has address line at most one Address_2
Examples:
Repair_Agent 'BCL District Teams' has address line Address_2 '147 Troy Street'
Repair_Agent 'BCL District Teams' has address line Address_2 'Te Rapa'
Repair_Agent 'Clear Communications' has address line Address_2 '147 Troy Street'

26. Repair_Agent is faxed on Fax
Each Repair_Agent is faxed on at most one Fax
Examples:
Repair_Agent 'BCL District Teams' is faxed on Fax '0-6-4569632'
Repair_Agent 'BCL District Teams' is faxed on Fax '0-4-9632587'
Repair_Agent 'Clear Communications' is faxed on Fax '0-6-4569632'

27. Repair_Agent is located in Town
Each Repair_Agent is located in at most one Town
Examples:
Repair_Agent 'BCL District Teams' is located in Town 'Wellington'
Repair_Agent 'BCL District Teams' is located in Town 'Auckland'
Repair_Agent 'Clear Communications' is located in Town 'Wellington'

28. Serial_Number belongs to Manufacturers_Name
Each Serial_Number belongs to at most one Manufacturers_Name
Examples:
Serial_Number '300014' belongs to Manufacturers_Name 'NEC'
Serial_Number '300014' belongs to Manufacturers_Name 'Marconi'
Serial_Number '564654' belongs to Manufacturers_Name 'NEC'

29. Serial_Number belongs to Model
Each Serial_Number belongs to at most one Model
Examples:
Serial_Number '300014' belongs to Model '71F2'
Serial_Number '300014' belongs to Model 'CTR122'
Serial_Number '564512' belongs to Model '71F2'

30. Serial_Number belongs to Part_Number
Each Serial_Number belongs to at most one Part_Number
Examples:
Serial_Number '300014' belongs to Part_Number '3612A'
Serial_Number '300014' belongs to Part_Number '80798-M1'
Serial_Number '564654' belongs to Part_Number '3612A'

31. Serial_Number equates to Manufacturers_Serial
Each Serial_Number equates to at most one Manufacturers_Serial
Examples:
Serial_Number '300014' equates to Manufacturers_Serial 'N/A'
Serial_Number '300014' equates to Manufacturers_Serial '6546466666'
Serial_Number '654654' equates to Manufacturers_Serial 'N/A'

32. Serial_Number has Bar_Code
Each Serial_Number has at most one Bar_Code
Examples:
Serial_Number '300014' has Bar_Code '4563548798764'
Serial_Number '300014' has Bar_Code '1245798764645'
Serial_Number '564314' has Bar_Code '4563548798764'

33. Serial_Number has been located in Past_Locations
Each Serial_Number has been located in at most one Past_Locations
Examples:
Serial_Number '000030000000' has been located in Past_Locations '22.3.95 MUB'
Serial_Number '000030000000' has been located in Past_Locations '1.4.95 WTE'
Serial_Number '000030120001' has been located in Past_Locations '22.3.95 MUB'

34. Serial_Number has been located in Past_Locations
Each Serial_Number has been located in at most one Past_Locations
Examples:
Serial_Number '300014' has been located in Past_Locations 'OTA MF20RX'
Serial_Number '300014' has been located in Past_Locations 'WTE MF27RX'
Serial_Number '564654' has been located in Past_Locations 'OTA MF20RX'

35. Serial_Number has history History
Each Serial_Number has history at most one History
Examples:
Serial_Number '000030000000' has history History '4.7.94 AS'
Serial_Number '000030000000' has history History '2.3.94 MO'
Serial_Number '000030120024' has history History '4.7.94 AS'

36. Serial_Number has history of History
Each Serial_Number has history of at most one History
Examples:
Serial_Number '300014' has history of History '2.3.94 AS'
Serial_Number '300014' has history of History '14.6.94 MO'
Serial_Number '564321' has history of History '2.3.94 AS'

37. Serial_Number has Serial
Each Serial_Number has at most one Serial
Examples:
Serial_Number '000030000000' has Serial '0'
Serial_Number '000030000000' has Serial '5'
Serial_Number '000030120001' has Serial '0'

38. Serial_Number has status of Status
Each Serial_Number has status of at most one Status
Examples:
Serial_Number '300014' has status of Status 'Suspect'
Serial_Number '300014' has status of Status 'In-Use'
Serial_Number '565464' has status of Status 'Suspect'

39. Serial_Number has status of Status
Each Serial_Number has status of at most one Status
Examples:
Serial_Number '000030000000' has status of Status 'Suspect'
Serial_Number '000030000000' has status of Status 'In-Use'
Serial_Number '000030120001' has status of Status 'Suspect'

40. Serial_Number has type Type
Each Serial_Number has type at most one Type
Examples:
Serial_Number '300014' has type Type 'Mux'
Serial_Number '300014' has type Type 'IF Filter'
Serial_Number '564632' has type Type 'Mux'

41. Serial_Number has type Type
Each Serial_Number has type at most one Type
Examples:
Serial_Number '300014' has type Type 'Mux'
Serial_Number '300014' has type Type 'IF Filter'
Serial_Number '564324' has type Type 'Mux'

42. Serial_Number is also identified by Manufacturers_Serial

Each Serial_Number is also identified by at most one Manufacturers_Serial

Examples:

Serial_Number '000030000000' is also identified by Manufacturers_Serial 'N/A'

Serial_Number '000030000000' is also identified by Manufacturers_Serial '123456789'

Serial_Number '000030120001' is also identified by Manufacturers_Serial 'N/A'

43. Serial_Number is identified as Module

Each Serial_Number is identified as at most one Module

Examples:

Serial_Number '000030000000' is identified as Module '0003000'

Serial_Number '000030000000' is identified as Module '0003006'

Serial_Number '000030120002' is identified as Module '0003000'

44. Serial_Number is located at Location

Each Serial_Number is located at at most one Location

Examples:

Serial_Number '300014' is located at Location 'MUB Store 1'

Serial_Number '300014' is located at Location 'WTE SPRVSR'

Serial_Number '564324' is located at Location 'MUB Store 1'

45. Serial_Number is located at Location which has status of Status and is altered on Date

Each Serial_Number, Location, Status, Date combination is unique

Examples:

Serial_Number '020020100002' is located at Location 'WTE DMRLDR' which has status of Status 'FAULTY' and is altered on Date '30.10.94 8:26'

Serial_Number '020020100002' is located at Location 'WTE DMRLDR' which has status of Status 'FAULTY' and is altered on Date '2.3.94 9:30'

Serial_Number '020020100002' is located at Location 'WTE DMRLDR' which has status of Status 'AVAILABLE' and is altered on Date '30.10.94 8:26'

Serial_Number '020020100002' is located at Location 'PNP DMRLDR' which has status of Status 'FAULTY' and is altered on Date '30.10.94 8:26'

Serial_Number '020020060000' is located at Location 'WTE DMRLDR' which has status of Status 'FAULTY' and is altered on Date '30.10.94 8:26'

Serial_Number '020020100002' is located at Location 'WTE DMRLDR' which has status of Status 'FAULTY' and is altered on Date '30.1.94 8:26'

Comment: "This refers to table "UPDATE". This table is downloaded from the barcode scanner which is used to track a module's movements. The same serial number can be contained with many different locations, status and dates as it changes location etc. Each line in the table is an hostorical statement which had I written the function, it would be read, and used to update the Table "Modules" and then deleted".

46. Serial_Number is located in Location

Each Serial_Number is located in at most one Location

Examples:

Serial_Number '000030000000' is located in Location 'MUB Store 1'

Serial_Number '000030000000' is located in Location 'WTE SPRVSR'

Serial_Number '000030120001' is located in Location 'MUB Store 1'

47. Serial_Number is owned by Owner

Each Serial_Number is owned by at most one Owner

Examples:

Serial_Number '300014' is owned by Owner 'BCL'

Serial_Number '300014' is owned by Owner 'Clear Communications'

Serial_Number '654551' is owned by Owner 'BCL'

48. Serial_Number is owned by Owner
Each Serial_Number is owned by at most one Owner
Examples:
Serial_Number '000030000000' is owned by Owner 'BCL'
Serial_Number '000030000000' is owned by Owner 'TVNZ'
Serial_Number '000030120001' is owned by Owner 'BCL'

49. Serial_Number is used for Usage
Each Serial_Number is used for at most one Usage
Examples:
Serial_Number '300014' is used for Usage '00'
Serial_Number '300014' is used for Usage '12'
Serial_Number '564346' is used for Usage '00'

50. Serial_Number print label
Examples:
Serial_Number '000030000000' print label
Serial_Number '000030120001' print label

51. Serial_NumberLocationStatusDat has Comment
Each Serial_NumberLocationStatusDat has zero or more Comment and
Each Comment has zero or more Serial_NumberLocationStatusDat that has it
Examples:
Serial_NumberLocationStatusDat '020020100002,WTE DMRLDR,FAULTY,30.1.94 8:26' has Comment 'Burnt'
Serial_NumberLocationStatusDat '020020100002,WTE DMRLDR,FAULTY,30.1.94 8:26' has Comment 'Perfect condition'
Serial_NumberLocationStatusDat '020020100002,WTE DMRLDR,AVAILABLE,30.1.94 8:26' has Comment 'Burnt'
Comment "I believe that this table was created when the data from the bar code reader was loaded incorrectly".

52. Site is named as Site_Name
Each Site is named as at most one Site_Name
Examples:
Site 'ABF' is named as Site_Name 'Abbotsford'
Site 'ABF' is named as Site_Name 'Acacia Bay'
Site 'ACB' is named as Site_Name 'Abbotsford'

53. Site is the responsibility of Team
Each Site is the responsibility of at most one Team
Examples:
Site 'ABF' is the responsibility of Team 'OTB'
Site 'ABF' is the responsibility of Team 'BPB'
Site 'ACB' is the responsibility of Team 'OTB'

54. Team has address 1 Address_1
Each Team has address 1 at most one Address_1
Examples:
Team 'AKB' has address 1 Address_1 '541 Scenic Drive'
Team 'AKB' has address 1 Address_1 'Level 3 BCL House'
Team 'GME' has address 1 Address_1 '541 Scenic Drive'

55. Team has address 2 Address_2
Each Team has address 2 at most one Address_2
Examples:
Team 'AKB' has address 2 Address_2 'Henderson'
Team 'AKB' has address 2 Address_2 '147 Tory Street'
Team 'GME' has address 2 Address_2 'Henderson'

56. Team is based in Town
Each Team is based in at most one Town
Examples:
Team 'AKB' is based in Town 'Auckland'
Team 'AKB' is based in Town 'Wellington'
Team 'GME' is based in Town 'Auckland'

57. Team is contacted on Fax
Each Team is contacted on at most one Fax
Examples:
Team 'AKB' is contacted on Fax '0-9-8149214'
Team 'AKB' is contacted on Fax '0-4-3826094'
Team 'GME' is contacted on Fax '0-9-8149214'

58. Team is contacted on Phone
Each Team is contacted on at most one Phone
Examples:
Team 'AKB' is contacted on Phone '0-9-8149208'
Team 'AKB' is contacted on Phone '0-4-3826000'
Team 'GME' is contacted on Phone '0-9-8149208'

59. Team is managed by Manager
Each Team is managed by at most one Manager
Examples:
Team 'AKB' is managed by Manager 'District Manager'
Team 'AKB' is managed by Manager 'G M Engineering'
Team 'GME' is managed by Manager 'District Manager'

60. Team is named as Teams_name
Each Team is named as at most one Teams_name
Examples:
Team 'AKB' is named as Teams_name 'Auckland District'
Team 'AKB' is named as Teams_name 'Engineering'
Team 'GME' is named as Teams_name 'Auckland District'

61. Team primarily maintains Site with usual Repair_Agent
Each Team, Site, Repair_Agent combination is unique
Examples:
Team 'MUB' primarily maintains Site 'WTE' with usual Repair_Agent 'BCL'
Team 'MUB' primarily maintains Site 'WTE' with usual Repair_Agent 'Clear'
Team 'MUB' primarily maintains Site 'ABF' with usual Repair_Agent 'BCL'
Team 'OTB' primarily maintains Site 'WTE' with usual Repair_Agent 'BCL'
Comment: "This refers to the table "defaults". It just sets the defaults of some fields in some forms".

62. Usage is described by Usage_description
Each Usage is described by at most one Usage_description
Examples:
Usage '00' is described by Usage_description 'Analogue Microwave'
Usage '00' is described by Usage_description 'TV Broadcast'
Usage '01' is described by Usage_description 'Analogue Microwave'

Appendix Eight *Product Acknowledgments*

Access® and Windows™ are registered trademarks of Microsoft®.

Paradox™ and DBase™ are registered trademarks of Borland.

Infomodeler is a registered trademark of Asymetrix.

Ontos™ is a registered trademark of ONTOS Inc.

Q&A™ is a registered trademark of Symantec Corporation.

FoxPro® is a registered trademark of Microsoft®

Informix™ is a registered trademark of Informix Software Inc.

Pick® is a registered trademark of Pick Systems Inc.

Advanced Revelation™ is a registered trade mark of Revelation Technologies.

SECTION NINE

REFERENCES

References

Astrahan, M. M. (1976). System R: a relational approach to database management. In ACM transactions on database systems. Vol 1 pp. 97-137.

Atkins, C.F. (1995). Prescription or description: some observations on the conceptual modelling process. Massey University working paper number 8-1995.

Atzeni, P., Batini, C., Lenzerini, M., and Villanelli, F. (1981). INCOD: A system for conceptual design of data and transactions in the entity relationship model. In Entity-relationship approach to information modelling and analysis. ER Institute, Saugas, California.

Bachman, C. W. (1969). Data structure diagrams. Database. Vol 1, No. 2, pp. 4-10.

Bachman, C. W (1988). A CASE for reverse engineering. Datamation. July pp. 49 - 56.

Balfour, A. (1988). Infoexec: the first practical implementation of a semantic database. Unisys. Europe-Africa division, December.

Batini, C., Ceri, S. And Navathe, S. B. (1992). Joint data and functional analysis. In Conceptual database design. USA: Benjamin Cummings.

Batra, D., Hoffer, J.A. and Bostrom, R.P. (1990). Comparing representations with relational and EER models. In Communications of the ACM. Vol 33, number 2. pp. 126 - 139.

Cattell, R.G.G. (1991). Object data management: object oriented and extended relational database systems. USA: Addison-Wesley.

Chamberlin, D.D., and Boyce, R.F. (1974). SEQUEL: A structured English query language. Proceedings of the ACM workshop on data description, access and control. New York, ACM.

Chang, C.L. (1976). DEDUCE - A deductive query language for relational databases. In Pattern recognition and artificial intelligence Ed Chen, C.H. New York: Academic Press.

Chen, P. (1976, March). The entity-relationship model - towards a unified view of data. In ACM transactions on database systems. Vol 1, No 1, pp 9-36.

Choobineh, J., Mannino, M. et al. (1988). An expert database design system based on the analysis of forms. IEEE Transactions on software engineering, Vol. 14, No. 2.

CODASYL (1971). CODASYL data base task group report in Conference on Data Systems Languages. New York: ACM.

Codd, E. F. (1969). Derivability, redundancy, and consistency of relations stored in large data banks. IBM Research report RJ599. San Jose: California.

Codd, E. F. (1970). A relational model of data for large shared data banks. Communications of the ACM. Vol 8, pp. 377-387.

Codd, E. F. (1971a). Normalised database structure: A brief tutorial. In Proceedings of the ACM SIGFIDET workshop on data description, access and control. pp. 1-17.

Codd, E. F. (1971b). A data base sub language founded on the relational calculus. In Proceedings of the ACM SIGFIDET workshop on data description, access and control. pp. 35-68.

Codd, E. F. (1972a). Further normalisation of the data base Relational Model. In Courant computer science symposia 6, data base systems. (Rustin, R., ed.), pp. 33-64. New York: Prentice-Hall.

Codd, E. F. (1972b). Relational completeness of data base sub languages. In Courant computer science symposia 6, data base systems. (Rustin, R., ed.), pp. 65-98. New York: Prentice-Hall.

Codd, E. F. (1974a). The relational approach to data base management: an overview. Third annual conference on computing systems. Austin: U.S.A.

Codd, E. F. (1974b). Recent Investigations in Relational Database Systems. Proceedings of IFIP. pp. 1017-1021.

Codd, E. F. (1979). Extending the database relational model to capture more meaning. ACM transactions on database systems. Vol. 4, No 4.

Codd, E. F. (1982). Relational database: a practical foundation for productivity. Communications of the ACM. Vol. 25, No. 2.

Codd, E. F. (1986). The twelve rules for relational DMBS. Technical report EFC-6. The Relational Institute. San Jose, California.

Codd, E. F. (1990). The relational model for database management. USA: Addison-Wesley.

Czarnik, B., Schuster, S. A., and Tsichritzis, D. C. (1975). ZETA: a relational data base management system. In Proceedings of the ACM Pacific '75. pp. 21-25.

Darke, P., and Shanks, G. (1995). Defining systems requirements: A critical assessment of the NIAM conceptual design procedure. Monash University Technical report.

Date, C. J. (1986a). Relational database selected writings. London: Addison-Wesley.

Date, C. J. (1986b). An introduction to database systems: volume one (4th edition). London: Addison-Wesley.

Date, C. J. (1993). An interview with Chris Date. DBMS magazine. March 1993

Davis, J.G. and Srinivasan, A. (1988). Incorporating user diversity into information systems assessment. In Information systems assessment. North-Holland. pp. 83-100.

Elmasri, R., Hevener, A., and Weeldreyer, J. (1985). The category concept: an extension to the entity-relationship model. Data knowledge engineering. Vol 1, number 1, pp. 75 - 116.

Fagin, R. (1977). A normal form for relational databases that is based on domains and keys. Technical report RJ2520. IBM Research Laboratory, San Jose, California.

Fagin, R. (1979). A normal form for relational databases that is based on domains and keys. Technical report RJ2520, IBM Research Laboratory, San Jose, California.

Fertuck, L. (1992). Systems analysis and design with CASE tools. USA: Wm. C. Brown

Flynn, D.J., and Fragoso-Diaz, O. (1993). Conceptual euromodelling: how do SSADM and MERISE compare? European journal of information systems. Vol2, number 3, pp. 169-183.

Gougen, J.A. (1992). The Dry and the Wet. In Information Systems Concepts: Improving the Understanding. Proc. IFIP working group 8.1 Conf. Alexandra, Egypt. pp. 1-17.

Halpin, T.A., and Orlowska, M.E. (1992). Fact oriented modelling for data analysis. Journal of Information Systems. 2, pp. 97-119.

Hammer, M.M., and McLeod, D.J. (1981). Database description with SDM: A semantic database model. ACM TODS. Vol 6, number 3.

Hammer, M.M., and McLeod, D.J. (1978). The semantic database model: A modelling mechanism for database applications. Proceedings of the ACM SIGMOD conference. Austin, Texas.

Hendry, D.G. and Green, T.R.G. (1994). Creating, comprehending and explaining spreadsheets: a cognitive interpretation of what discretionary users think of the spreadsheet model. International journal of human-computer studies. 40, pp. 1033-1065.

Hull, R. and King, R. (1987). Semantic database modelling: Survey, applications, and research issues. ACM computing surveys. Vol. 19, No 3, pp 201-260.

Johnson-Laird, P.N. (1983). Mental models. Cambridge University Press: UK.

Kent, W. (1983). A simple guide to five normal forms in relational database theory.

Lenzerini, M., and Santucci, G. (1983). Cardinality constraints in the entity-relationship model. In Davis, G., et al., (Eds) (1983) The entity-relationship approach to software engineering. Elsevier North-Holland: New York. pp 529 - 549.

Loosley, C., and Gane, C. (1989). The object of processing is data. InfoDB. Vol 4, number 4.

Ling, T. (1985). A normal form for entity-relationship diagrams. In proceedings of the 4th international conference on the entity-relationship approach (Chicago). IEEE Computer Society Press, Silver Springs, Md., pp. 24 - 35.

Maier, D., Ulman, J.D., and Vardi, M.Y. (1984). On the foundations of the universal relational model. In ACM transactions on database systems. Vol 9, number 2.

Mark, L. (1988). The binary relationship model. Institute of advanced computer studies. In Codd, E.F. (1990). The relational model for database management. USA: Addison-Wesley.

Markus, M. L. (1989). Case selection in a disconfirmatory case study. In Harvard business school research colloquium. Boston, MA: Harvard publishing.

Mattison, R. (1993). Understanding database management systems: an insider's guide to architectures, products and design. New York: McGraw-Hill

McCormack, J.I., Halpin, T.A., and Ritson, P.R. (1993). Automated mapping of conceptual schemas to relational schemas. Proceedings CAiSE '93. pp 432-448.

McWilliams, G. (1988). Users see a CASE advance in reverse engineering tools. Datamation.

Navathe, S., and Cheng, A. (1983). A methodology for database schema mapping from extended entity-relationship models into the hierarchical model. In Davis, G., et al., (Eds) (1983) the entity-relationship approach to software engineering. Elsevier North-Holland: New York.

Nijssen, G. M., and Halpin, T. A. (1989). Conceptual schema and relational database design a fact oriented approach. Prentice Hall: Sydney.

Olle, T. W. (1993). Data modelling and conceptual modelling: A comparative analysis of functionality and roles. Australian journal of information systems. Vol, pp. 46-57.

Oren, O. (1985). Integrity constraints in the conceptual schema language SYSDOC. In proceedings of the 4th international conference on the entity-relationship approach (Chicago). IEEE Computer Society Press, Silver Springs, Md., pp. 288 - 294.

Pirotte, A. And Wodon, P. (1977). A comprehensive formal query language for a relational database. R.A.I.R.O. Informatique / computer science. 11 number 2.

Premerlani W.J. and Blaha, M.R. (1994) An approach for reverse engineering of relational databases. Communications of the ACM. Vol 37, number 5 pp42 - 49.

Quang, P.T. (1986). MERISE: a French methodology for information systems analysis and design. Journal of systems management. March, pp. 21-24.

Ricardo, C. (1990). Database systems: principles, design and implementation. New York: Maxwell Macmillan.

Rock-Evans, R and Hales, K. (1990). Reverse Engineering: methods and tools. Ovum Ltd. Vol. 1 and 2.

Ryder, M. R. (1993). A CASE for relational data modelling. Unpublished honours report. Massey University.

Sabanis, N. and Darlison, A. (1992). Reverse engineering data using an integrated toolset. In Spurr, K., and Layzell, P. (Eds.). (1992). CASE current practice future prospects. Chichester: John Wiley & Sons.

Sakai, H. (1983). Entity relationship approach to logical database design. In Davis, G., et al., (Eds) (1983) the entity-relationship approach to software engineering. Elsevier North-Holland: New York. pp. 155 - 187.

Scheuermann, P., Scheffener, G., and Waber, H. (1980). Abstraction capabilities and invariant properties modelling within the entity-relationship approach. In Chen, P. (Ed) the entity-relationship approach to systems analysis and design. North-Holland: Amsterdam. pp. 121 - 140.

Simsion, G. (1994). Data modelling essentials: analysis, design and innovation. International Thompson Computer Press: USA.

Smith, J. M., and Smith, D.C.P. (1977). Database abstractions: aggregation and generalisation. ACM transactions on database systems. Vol 2, pp 105-133.

Sneed, H. M. and Jandrasics, G. (1988). Inverse transformation of software from code to specification. Proceedings of IEEE conference on software maintenance. Phoenix, Arizona.

Srinivasan, A. (1992). Autonomous application development using data modelling. Some research issues. Informatica. Vol 1, number 1. pp. 7-22.

Stonebraker, M. R., Wong, E. and Kreps, P. (1976). The design and implementation of INGRES. In ACM transactions on database systems. Vol 1, pp. 189-222.

Teory, T.J., Yang, D., and Fry, J.P. (1986). A logical design methodology for relational databases using the extended entity-relationship model. Computing surveys. Vol 18, number 2, June.

Todd, S. J. P. (1975). Peterlee relational test vehicle PRTV, a technical overview. IBM UK scientific centre report UKSC-0075. Peterlee, England.

Tsichritzis, D. C. and Lochovsky, F. H. (1976). Hierarchical data-base management: A survey. ACM computing surveys. Vol. 8, pp. 67-103.

Tsichritzis, D. C. and Lochovsky, F. H. (1982). Data models. Englewood Cliffs, N.J.: Prentice-Hall.

Webre, N. (1981). An extended entity-relationship model and its use on a defence project. In proceedings of the 2nd international conference on the entity-relationship approach (Washington, D.C.). North-Holland: Amsterdam. pp. 175 - 194.

X3H2, American National Standards Database Committee. (1985). Draft proposed American national standard database language SQL. Document X342-85-40. USA: ANSI.

Yin, R. K. (1989). Case study research: design and methods (2nd ed). London, UK: Sage Publications.

Zaniolo, C. (1986). Safety and compilation of nonrecursive horn clauses. In proceedings of the international conference of expert database systems. In Ricardo, C. (1990). Database systems: principles, design and implementation. New York: Maxwell Macmillan.

Zloof, M. M. (1975). Query by example. In Proceedings AFIPS NCC. Vol 44, pp. 431-438.