Expert Assistance for Database Design

by

Roger Allen Vasconcells

B.S., Kansas State University - Manhattan, 1980

A MASTER'S THESIS

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1987

Approved by:

[illegible]

Major Professor

TABLE OF CONTENTS

LIST OF FIGURES AND TABLES

ACKNOWLEDGMENTS

I wish to thank my advisor, Dr. Elizabeth Unger, for the understanding, assistance, and support she gave me with this thesis. As always, working with her has been both an honor and a pleasure.

The other members of my committee, Dr. Virgil Wallentine and Dr. David Schmidt also deserve thanks for their support and guidance.

I also wish to thank, Dr. Austin Melton, for serving in the place of Dr. Schmidt.

Special thanks to Dennis Reith for his assistance that provided for the typing of this thesis.

Special thanks also go to my family, Ross, Norma, Bart and Mary Jo, and my brother and best friend Ben, whose encouragement was always appreciated.

Finally and most importantly, I thank my lovely wife, Joyce, whose support, encouragement, assistance, and understanding throughout the history of this thesis allowed me to follow through with its completion.

Chapter 1

INTRODUCTION

Database management systems (DBMSs) have proven that they are powerful, efficient, cost effective, and highly productive tools in a large variety of areas. Because of growing interest in their applications the technology and development of DBMS have reached such intensity that various subfields have evolved. Database integrity and security, database design, data recovery, and data organization are some examples of this evolution. Although these subfields have caused a diversification of expertise in DBMSs, the knowledge necessary to create a significantly higher quality system has increased proportionately.

The database design subfield, one of the major activities of the system development process, requires difficult, complex, and time-consuming tasks. Problems arising from inadequate designs caused by vague specifications of organizational goals and requirements produce limited or useless databases not capable of adapting to change. High response time, high storage demands, inconsistent data, uncontrolled redundancy, and low user acceptance are all characteristics of a poorly designed DBMS. These problem-ridden systems often impede a DBMS's effectiveness as a data processing tool.

Many theories and practices have been applied to the design of DBMSs to reduce the possibility of creating a problem-ridden database. However, few efforts have produced methodologies that allow the design task to be supplemented with expertise from any source other than what the designer could provide. If the experience of many designers could be brought to bear on the design process, any designer given the ability to utilize this expertise, could reduce the possibility of creating an unusable database. This type of formalism would allow the designer freedom to concentrate on the design task itself and insure that a particular design tool's rules would be enforced, providing for the creation of a problem-free database.

Although a relatively young discipline in Computer Science, Artificial Intelligence (AI) has recently stimulated a great amount of research and interest. AI found its beginnings in studying problems associated with game-playing and theorem proving. With advancements in technology and research, vast amounts of knowledge could be stored allowing expansion of the applications of AI. These included the perception of vision and speech, natural language understanding, and specialized problem solvers, such as medical diagnosis and chemical analysis.

Much like database design, medical diagnosis and chemical analysis are tasks not routinely performed everyday and require vast amounts of specialized knowledge to function properly. Programs called Expert Systems (ES) were developed that could access and use large amounts of this type of domain-specific knowledge. ES technology, a proven tool in the field of AI, will allow an enhancement to the design phase of DBMS development by providing a means of accessing large amounts of knowledge specific to DBMS design.

A description of such an enhancement to the design of DBMSs will be presented in this thesis. A theoretical model combining the technology provided by Expert Systems and an existing DBMS design tool will be proposed. This Expert Assistant will provide both a means of storing the expertise of numerous designers and a method of applying this expertise to the design process.

Following a review of relevant literature regarding database design and Expert Systems, a description of how the Expert Assistant will enhance the design of DBMSs will be presented. A presentation of the particular DBMS design tool that will be enhanced is defined and demonstrated. The architecture that the Expert Assistant is based upon is discussed, and the taxonomy of the Expert Assistant will be defined preceding a step by step example of a minimal implementation. Finally, possible improvements, difficulties, and further research will be discussed.

Chapter 2

LITERATURE REVIEW

2.1 DATABASE DESIGN

One of the most crucial phases of database development is the design of an effective database structure. [Yao 85] describes the goal of database design as a process which organizes databases to facilitate effective processing and to involve various activities, beginning with definition of the problem and ending with a system implementation. Prior to investment in an implementation, the database design should allow for analysis of system correctness and performance [Wiederhold 83]. Producing a workable database that makes data available to users while maintaining data integrity, security, and minimized redundancy requires a design mechanism which presents the data in a usable format [Turk 85]. [Cardenas 79] proposed a definition that is conceptually easy to understand and provides for many, if not all, of the requirements mentioned in other database design formalisms.

> File and database design is the process of synthesizing the collection and associations of data to satisfy the information storage, retrieval, and reporting requirements of users cost-effectively, while meeting a number of constraints (not always mutually compatible) such as access time, flexibility of use, storage, security, auditing, and recovery. The design of databases is in actual practice usually an iterative process, often involving trial and error, just like the design of the information systems which the databases are intended to support. [Cardenas 79]

```
                        Step

                 /        1. Statement       of       requirements
                 |           (information       flows,        data
                 |           transformation,  reports,   queries,
                 |           performance criteria)
                 |
                 |        2. Logical    or   conceptual   database
Logical          |           structure   in   a given  information
Database        <            model
Design           |
                 |      / 3. a)  Database schema definition  via
                 |      |        the    schema    data    description
                 |      |        language.
                 |      |
                 |      |     b)  Subschema  definition  via  the
                 \      |        subschema data description language
                        |
                        |
                        | 4. Access   path  determination  (e.g.
Physical                |    secondary indexing)
Database               <
Design                  | 5. Mapping    and    representation   of
                        |    logical    data   on  physical    data
                        |    structures   (e.g.   Database    Task
                        |    Group [DBTG] areas)
                        |
                        | 6. Physical  layout of data on storage
                        |    devices available and determination
                        |    of   low  level   data   management
                        |    parameters (e.g. buffers, blocking,
                        \    device areas)


                        / 7. Actual   data   base   loading   and
Database                |    installation
Operation and          <
Reorganization          | 8. Tuning and retuning or redesign due
                        \    to changing requirements
```

Figure 2.1 The file and database design process. (Adapted from [Cardenas 79])

This definition provides for three major stages which are involved in designing a database:

1. Logical database design,
2. Physical database design, and
3. Database loading and operation.

These stages are composed of various steps encompassing the entire file and database design process (see Figure 2.1). [Turk 85] presents these steps in a business application viewpoint, and represent the same protocols. These stages have also been represented as a conceptual level and a organizational level [Wiederhold 83]. At the conceptual level, information from the user is dealt with, at the organizational level data representation and data processing are defined. The taxonomy of these steps within each stage are not always clear cut, but there usually exists a finite separation between the particular stages. Cardenas' step by step process of designing a database entails a simplistic description of database design and will be presented here to form a basic definition.

2.1.1 LOGICAL DATABASE DESIGN

As seen in Figure 2.1 the logical database design begins with identifying information needs within the scope of the users data, forming a conceptual database independent of any computer architecture. Included in this phase is criteria for expected performance, access time, storage requirements, future expectations, security needs, integrity rules, and justification of investment returns of the application.

The next step in the design is to convert this conceptual database into a logical database using a particular data model whether it be network, hierarchical, or relational. Whichever model is chosen, each have specific data structures that can be used to form a logical database that is equivalent to the conceptual database. The three choices of data models and database structures are assisted by the following criteria:

1. The contents of the database,
2. The characteristics of the users' data accessing requirements,
3. The characteristics of the particular database structure and DBMS used, and
4. The characteristics of the hardware used.

Step three begins with the definition of the logical description of the global database, the database schema. The schema includes the definition of the names and data types of each field of all data structures defined in the conceptual database as well as the relationship of the linkage between any two data files. This description is defined in the Data Definition Language (DDL) supported by the particular data model chosen. Although completed at the logical design level the physical design does influence some of the decisions made in this step since each data model has a unique method of locating a particular record occurrence. Each data model presents its own restrictions and complications occurring with their schema DDL and should be adhered to closely.

2.1.2 PHYSICAL DATABASE DESIGN

Physical database designs' objective is to choose among the many file structures and methods of linking data files into a database, and use the options available with the chosen data model for tuning to optimum performance [Merrett 84]. There is a definite separation between logical and physical database design, however it is not always clear between various steps within the physical design stage. [Cardenas 79] begins this design phase by following the schema definition with the definition of the subschema using the subschema DDL (Figure 2.1 step 3a), usually a variation of the schema DDL excluding physical structure details. This is a logical description of a subset of the database, and any number of subschemas may be defined over a schema. Similar to defining the schema the subschema definition includes fields, record types, and database relationships assigned to a particular users database application.

Step four (as shown in Figure 2.1) initiates the physical design by taking the logical design and defining access paths into the database and between records. These paths are dependent upon the particular random access method supported by the computer architecture the implemented database will reside on. Once the random method is chosen various alternatives based on what data model is used are evaluated, e.g. if the data model chosen allows a choice of secondary indexing methods, then the decision of which to use is completed in this step.

Step five describes how various record types will be grouped in order to reduce overall access time, reduce storage fragmentation, and enforce integrity and recovery constraints. Because of the complexity of the access path determination in step four and the physical layout of data in step six, the processes described in this step may be indistinguishable. This is true for those data models that do not provide a clear separation between the logical and physical data structures used. Some dynamic storage mechanisms require an overflow definition which include how the overflow areas will be used for any particular need and what type of overflow is needed.

The sixth step includes all the specifications of requirements needed to allow the actual loading of the database on to the particular installation. Included in this step is the definition of physically mapping the record types on storage devices based on system dependent physical parameters. This includes what hardware addresses will be used to hold overflow and data areas, the type of external storage device to be used, blocking factors, data field formats, data compression schemes, etc. Each data model has its own mechanisms and conventions for designating standard formats of data and a language for communicating these physical details.

2.1.3 DATABASE LOADING AND OPERATION

Database loading starts the third phase of database design and the seventh step as shown in Figure 2.1. Various utilities are usually provided by each particular system to facilitate loading the database on a particular installation. This process can be complicated by the physical limitations mentioned in step six. For databases that are complex this step can be a time consuming task when physical limitations are overly complicated.

Step eight in Figure 2.1, covers tuning, operation, and reorganization of the loaded database. Although the reason for database design is to formulate a database that will perform satisfactorily, various testing, tuning, and redesigning steps are needed to produce a substantial end product. Since there are so many parameters in the design of a database and the interrelationship between them can be so complex, the eighth step is defined to provide for any need to reload or redesign the database to make it viable to the user.

2.2 DATABASE DESIGN AIDS

The technology of DBMSs have become accepted and used in a wide variety of applications. [Fry 78] maintained that the database designer was faced with the problem of not which database system to use but how to use it effectively. This led to the development of tools to assist the database designer in each phase of a DBMS's

```
             |
             |      HIPO •
  L D T      |      SADT                 CADES
  O E O      |      DFD                  CADIS
  G S O      |      -----                CASCADE
  I I L      |      AUXCO ⁺
  C G S      |      ADS                  PSL/PSA
  A N        |      ARDI
  L          |
             |
             |_________________________________________
             |      Manual       |   Computer Aided    |

                           Technique
```

Figure 2.2 Categorization of Logical Database Design Tools
• Formal Language Method
⁺ Graphically-Oriented Method

life cycle, proving that the usefulness and responsiveness of a DBMS could be enhanced.

2.2.1 DESIGN TOOLS

The initial step of any logical database design is to specify a definition of the users requirements. Figure 2.2 lists several logical database design tools discussed by [Kahn 85], and categorizes them into their area of assistance. As Figure 2.2 shows, there exists two techniques to assist in the requirements design process: manual techniques and computer-aided techniques. Both of these techniques should provide the designer with assistance in what the system should eventually do, what activities will be required, the data necessary to implement the system, and other requirements typical of the database users organization.

The manual techniques are categorized into two different types of tools, those that use some natural or formal language to produce tables and charts, and those that are graph oriented using pictorial representations of requirements and have little or no language augmented. AUXCO divides each phase of the requirements design into many minute activities, each explaining appropriate techniques, denotation of decision points, and positive alternatives. Developed by NCR, the Accurately Defined System (ADS) is a forms based backward-forward analysis technique, and provides a mechanism that determines and depicts the information that flows through a database. A four phase system development process: Analysis, Requirements determination, Design and development, Implementation and valuation (ARDI) [Hartman 68], is a planning network representing each design phase which may be further divided into sub-steps.

The Hierarchical Input Process Output (HIPO) technique developed by IBM [Jones 75] and [Katzan 79] is a top-down process-oriented approach to database design. HIPO is intended for use by software designers and programmers concentrating on process definition and hierarchically decomposes the database into processes and modules, called functions. Using an input process output diagram formalism each function is documented as a process along with its inputs and outputs. SofTech, Inc. developed the Structured Analysis and Design Technique (SADT) to provide assistance in performing system analysis and design in requirements and logical design phases of DBMS development. Methods used with SADT allow for top-down structured thinking, requirements documentation, project planning, managing, and evolution. DFD uses Data Flow Diagrams to describe a database system using directed graphs and is based on work originating with [Gane 77] and [DeMarco 84]. DFD is a top-down technique that can be utilized with either a process-oriented, a data-oriented, or a backward-forward approach and is used initially to diagram a systems' information flow. Then it decomposes each diagram into a hierarchy of directed graphs that distinguish between data entering and leaving a database system.

In most cases, computer-aided techniques are an implementation of existing manual techniques providing both a formal language for specifying requirements and a method for generating standard reports. These reports range from narratives, to lists and tables, and finally to pictorial presentations. Computer-Aided Design of Information Systems (CADIS) developed by the Department of Information Processing at the Royal Institute of Technology of Stockholm, Sweden [Bubenko 72], is a tool for database documentation analysis. CADIS includes an information storage and retrieval system, a binary language syntax editor, and a report generator. The Computer-Aided Systems Construction and Documentation Environment (CASCADE) from the University of Trondheim, Norway, includes a graphic technique for describing database design and database requirements. Using CASCADE, the system design process is automatically documented with flowcharts and lists. The Computer-Aided Design and Evaluation system (CADES) from International Computers Limited, spans the gap between design and implementation in the initial design process. Using a data-driven approach, CADES automatically generates implementation code and code that tests the specifications of the design. PSL/PSA the Problem Statement Language and Problem Statement Analyzer developed by the University of Michigan as part of the ISDOS (Information System Design and Optimization System) project, is a computer aided system analysis and logical design formalism. PSL allows the designer to state a users requirements in a human-machine readable form, allowing the user to specify what will be required and not so much as how the requirements will be satisfied. PSA takes the analyzed components of PSL and places them into a computerized database. PSA accesses these components to modify and update them and uses them to generate standard reports.

Conversion of the users requirements into a schema definition or conceptual view is a major stumbling block in the design process. As an answer to this problem [Ruoff 84] produced an information modeling technique called IDEF1 or ICAM Definition Method Version 1 (ICAM is an acronym for Integrated Computer-Aided Manufacturing). IDEF1 confronts areas of development that are considered most difficult to establish, i.e. information objects, relationships between objects, and their properties. The Information Resource Specification and Design Language (IRSDL), is a tool for specifying the requirements of logical database design [Konsynski 79]. Geared towards the non-specialist designer, IRSDL provides specifications for user documentation, user views of the conceptual schema, and reorganization of the conceptual schema.

Tying the logical database design to the hardware that will eventually support the system, as might be expected the physical design phase also has been supplied with design assistance tools. The SEmantic DAtabase COnstructor (SEDACO) is used to implement logical schemas and provides protection of low-level data structure issues from the designer and has the ability to efficiently maintain consistency within complex semantic databases [Farmer 84]. [Orlando 85] proposed two integrated tools which are used in the physical database design phase. System EOS predicts database performance based on various evaluation models, allowing the precise prediction of the application workload and the performance behavior of a completed database. System EROS is based on optimization and uses the evaluation models of System EOS to estimate the cost of other implementation solutions. Many design tools assisting with the physical structure of a database are supplied by the particular implementation the user has purchased. ESTIMATE, a utility program for CDC's SCOPE operating system takes the record and key descriptions and produces suggested sizes for data and index blocks as well as memory buffers. System 2000 from MRI Systems Corporation provides statistical assistance to the database designer by supplying counters which monitor accumulated CPU time, real time, and input output operations for every data set being used.

Database design tools that provide assistance to the database designer in more than one phase of the database design process have also been developed. [Bragger 84] described a data definition system called GAMBIT, which produces a definition of static or physical data structures, a description of semantic integrity constraints by using a full programming language, and a data description language. GAMBIT assists the database designer by graphically representing the schemas on a screen which can be modified. The DATAID project [Albano 85] produced design methodologies covering all phases of the database design process, including interactive tools for logical data analysis, prototyping to reduce operation and maintenance costs, and various other automated tools that support design techniques. [Komorowski 84] presented the advantages that are obtained by using PROLOG as a software prototyping tool for DBMSs. He showed that by using PROLOG a database of parsing trees could be developed as well as natural language interfaces.

2.3 ENTITY-RELATIONSHIP MODEL

The variety of database design tools reviewed span a wide area of application as well as the numerous theoretical models that they are based on. Presenting a technique that is simple and appears to many people to be quite natural [Chen 85] stated that the reason for his Entity Relationship model being so simple was:

> that it focuses on a fundamental issue of database design: what does it represent?

He went on to explain that a database is an extension of the environment in which it is used and that real world domains can be represented in the form of entities, relationships, and the attributes that pertain to them. The semantic inadequacies of the relational model, the difficulties the network model had with achieving data independence, and the unnatural data characteristics of the entity set (hierarchical) model led [Chen 76] to the proposal of the Entity-Relationship (ER) model. The E-R model encompasses most of the advantages of the three previously mentioned models and included a more natural view of the real world. The original E-R model incorporated some of the significant semantic information about the real world and could achieve a high amount of data independence based on set theory and relation theory.

Entity-Relationship Diagrams (ERD) were defined to allow the graphical representation of entities, relationships, and their respective attributes. A designer using ERDs can define all entities and their relationships whether they be 1:1, 1:N or M:N and maintain semantic integrity. Stipulations for data description and data manipulation through rules for insertion, deletion, and updating of values were defined in the original E-R model. After a users data has been defined in a logical schema created from the ERDs, Chen showed how each of the other three database models could be derived from the E-R model.

In order to maintain a database design truly reflective of the environment in which it will be used, [Chen 77] modified the E-R model by introducing an intermediate step in the design process. This step incorporated the enterprise view of data and provided for the following advantages:

1. The enterprise schema is easier to understand than a user schema since the former does not have the restrictions of the underlying database management system;

2. The enterprise schema is more stable than the user schema, since some types of changes in the user schema may not require any change in the enterprise schema. If the enterprise schema needs to be changed to reflect the changes in the enterprise environment, the changes can be performed easily since efficiency and storage issues are not considered.

[Chen 82] described how to use the E-R model by presenting a step by step process of designing an order entry database. In [Chen 84] further modifications to the E-R model were introduced to facilitate easier use of the model in a wider variety of areas by classifying the model into two categories. The Generalized [N-ary] Entity-Relationship model (GERM), allows relationships to be defined on more than two entities, and the Binary Entity-Relationship model (BERM) allows at most two entities to be involved in a relationship. This classification allowed for many significant effects on modeling and analysis. A designer could use their favorite model to define a system then convert it to other models that might be more easily understood for presentations, or it could increase the possibility of proving that two E-R models

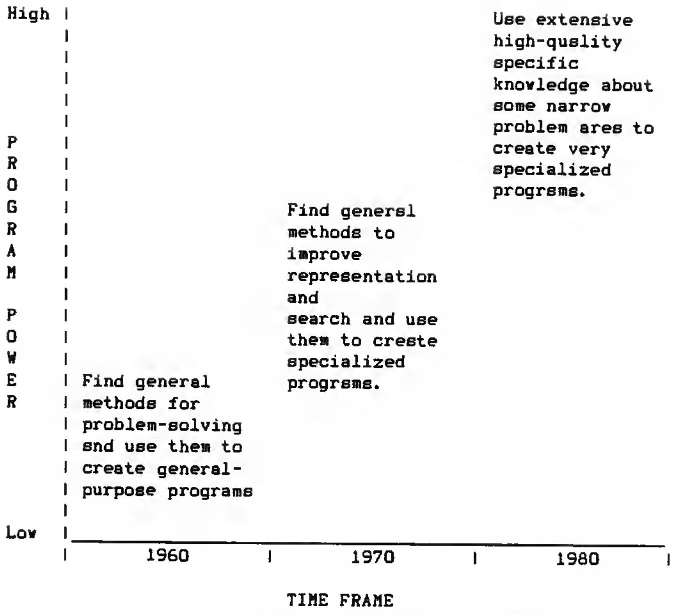


Figure 2.3 The shifting focus of AI research.
(Adapted from [Waterman 86])

are equivalent. [Chen 84] has prosed an E-R algebra for BERM which included directional relationships that would be useful in designing query languages for DBMSs based on the E-R model. The most current modification of the E-R model removed attributes from a relationship and created a composite entity (an entity formed by other entities) which would obtain these attributes [Chen 85]. Removing attributes from relationships clears the ERDs, assisting in the overall simplification of the E-R model.

2.4 EXPERT SYSTEMS

The primary goal of Artificial Intelligence has always been to provide a means by which a computer program could solve a problem in a manner that would be considered intelligent by humans. For the last twenty years Expert Systems have been striving for acceptance as well as a definition with this field. Figure 2.3 graphically represents how Expert Systems have figured historically in Artificial Intelligence.

During the decade of the 1960's scientists strove to create programs which could solve problems with a general area of application. These projects soon became frugal at best. It seems that the more general these programs were made in their problem solving capability, the more inept they became in solving any particular problem. This lead to concentrating efforts on applying these programs to specialized problems. In the early seventies a concentrated effort on how to represent and search through the knowledge needed to solve these specialized problems provided some success but there were still no major breakthroughs. In the late seventies the realization came that the true power of a problem solver was not found in its solving abilities but in the knowledge it possesses. Stated simply:

> To make a program intelligent, provide it with lots of high quality specific knowledge about some problem area [Waterman 86].

```
                EXPERT SYSTEM
      ___________________________
     |                           |
     |     KNOWLEDGE  BASE       |
     |    (Domain  Knowledge)    |
     |   --------------------    |
     |  |       FACTS        |   |
     |   --------------------    |
     |  |       RULES        |   |
     |   --------------------    |
     |           / \             |
     |            |              |
     |           \ /             |
     |   --------------------    |
     |  |    INTERPRETER     |   |
     |   --------------------    |
     |  |     SCHEDULER      |   |
     |   --------------------    |
     |                           |
     |     INFERENCE ENGINE      |
     |        (General           |
     |     problem-solving       |
     |       knowledge)          |
     |___________________________|
```

Figure 2.4 The structure of an expert system. [Waterman 86]

Because of this realization, the problem solvers evolved into programs that were "expert" in a specific domain and Expert Systems became a viable tool in the problem solving paradigm.

At one time the ability to create an Expert System was considered to be artistic and not scientific. The work of numerous Artificial Intelligence scientists, brought together in <u>Building Expert Systems</u> [Hayes-Roth 83], provided for a better understanding and a clear definition of designing Expert Systems. Knowledge Engineering is generally accepted as the definition for building Expert Systems. The process is best described as coordinating

Category	Problem Addressed
Interpretation	Inferring situation descriptions from sensor data
Prediction	Inferring likely consequences of given situations
Diagnosis	Inferring system malfunctions from observables
Design	Configuring objects under constraints
Planning	Designing actions
Monitoring	Comparing observations to plan vulnerabilities
Debugging	Prescribing remedies for malfunctions
Repair	Prescribing a plan to administer a prescribed remedy
Instruction	Diagnosing, debugging, and repairing student behavior
Control	Interpreting, predicting, repairing, and monitoring system behaviors

Table 2.1 Generic categories of knowledge engineering applications.
(Adapted from [Hayes-Roth 83])

interaction between the knowledge engineer and one or more human experts in a particular problem area. The Knowledge Engineer retrieves the knowledge necessary to solve a problem and uses this to build an Expert System resulting in a problem-solver with capabilities approaching that of humans.

2.4.1 TAXONOMY OF EXPERT SYSTEMS

Expert Systems can be categorized by the fields in which they are used, and the by types of problems to which they are applied. The

Agriculture	Manufacturing
Chemistry	mathematics
Computer Systems	Medicine
Electronica	Meteorology
Engineering	Military Science
Geology	Physica
Information Management	Proceaa Control
Law	Space Technology

Table 2.2 Application areaa for expert aystema. [Waterman 86]

accepted atructure for all Expert Syatem architecturea can be seen in Figure 2.4, any variation in Expert Syatem applicationa can be in the method of acceaaing the knowledge in the knowledge baae, how the knowledge ia interpreted, interfacing with its uaers, etc. Table 2.1 aummarizea the typea of Expert Syatem applicationa that exiat.

2.4.2 EXAMPLES OF EXPERT SYSTEMS

The quantity of Expert System applicationa ia proportionate to the demand for their use. Table 2.2 demonatratea thia by liating various fielda that aupply a need for their uae. The Expert Systema uaed in the Computer Syatema field discuaaed by [Waterman 86] and illuatrated in Figure 2.5, ahowa a wide range of theae applicationa.

XCON developed by Digital Equipment Corporation and Carnegie-Mellon University in the late 1970's ia one of the firat aucceaaful applicationa in the computer ayatema area. XCON configures VAX

```
 ________________________
|                        |
| Computer   Systems     |
|________________________|
            |
            |-- Prediction ----------< PTRANS
            |
            |
            |                    ----<  CRIB
            |-- Diagnosis   ----<
            |                    ----<  PTRANS
            |
            |
            |-- Design      ----------<  XCON
            |
            |
            |-- Planning    ----------<  PTRANS
            |
            |
            |                    ----<  PTRANS
            |-- Monitoring  ----<
            |                    ----<  YES/MVS
            |
            |
            |                    ----<  PTRANS
            |-- Debugging   ----<
            |                    ----<  TIMM/Tuner
            |
            |
            |                    ----<  PTRANS
            '-- Control     ----<
                                 ----<  YES/MVS
```

Figure 2.5 Selected expert systems in the computer systems domain. (Adapted from [Waterman 86])

11/780 computer systems by combining customer order information, site architecture, and known system physical limitations. CRIB, developed by International Computers Limited, the Research and Advanced Development Centre, and Brunel University, assists engineers in searching and discovering faults in computer systems in the field. Simple english is used to input the faults discovered into the system and CRIB matches this against a knowledge of faults that have occurred previously. Developed by Digital Equipment Corporation and Carnegie-Mellon University, PTRANS aids in the control of the manufacture and distribution of Digital Equipment Corporation's computer systems. This is accomplished by using customer order information and manufacturing information to develop plans for building and debugging ordered systems. YES/MVS, developed by IBM, monitors and controls Multiple Virtual Storage (MVS) operating systems to assist computer operators. TIMM/TUNER a commercial system developed by General Research Corporation assists with the tuning of VAX/VMS computer systems in order to reduce performance problems that arise in a constantly changing computer environment.

2.5 RELEVANT CURRENT WORK

Much attention and interest has been focused on the E-R model. Conferences centered on the E-R model have produced many insights into its application, possible improvements, and the study of what effects other computer science disciplines might have on it. The following sections focus on these areas concerning the E-R model by reviewing current literature covering them.

2.5.1 ENHANCEMENTS TO THE ENTITY-RELATIONSHIP MODEL

The original E-R model proposed by [Chen 76] was designed to assist in logical database design and did not present a formalism for converting the E-R model into other database models. Several attempts have been made to form a tranalation formalism, [Chung 83], [Dumpala 83], [Hwang 83], [Melkanoff 79], and [Morgenatern 83], that are at beat heuriatic in their formalized guidelines. They do not include aome of the E-R concepta auch as compoaite attributes, weak entity typea, recursive relationship sets, and weak relationahip aeta.

An anawer to these aimpliatic attempta at E-R converaion waa furniahed by [Ling 85] who propoaed a normal form for ERDs that would obtain the following objectivea:

1. to capture and preaerve all the aemantica of the real world of a database which can be expreaaed in terma of functional, multivalued, and join dependenciea, by repreaenting them explicitly in the E-R diagram.

2. to enaure that all the relationahipa repreaented in the E-R diagram are non-redundant, i.e. none of the relationahipa can be derived from other relationahipa.

3. to ensure that all the relationa tranalated from the E-R diagram are in good normal form, either in 3NF or 5NF.

Thia normal form allowed for composite attributea, multivalued attributea, and apecial typea of relationship seta. An algorithm ia uaed that translates an ERD, in normal form, to a set of relationa, which are either in 3NF (third normal form) or 5NF (fifth normal form).

Another formalism which modifiea the original E-R model [Chen 76], waa created by [Brady 85]. A univeraal relation aaaumption waa presented that removed logical navigation of the databaae. This asaumption provides logical and physical data independence and a very simple data apecification aystem. The conceptual database created, using the E-R approach that incorporates the universal relation assumption, will have a direct translation into a relational database. [Brady 85] furthered the use of the assumption by refining it to a E-R univeraal relation aaaumption which alleviates some of the limitationa of the original asaumption, i.e. limited applications and distortion of database design. The new version basea ita reliability on the fact that it uaes rules that rely on a widely accepted and atandard model of the real world.

[Cazin 85] introduced the F1 formaliam which allowa the description of the conceptual schema of a database. This formalism was deaigned to deacribe and use an Information System (IS) in a Software Engineering Environment (SEE). The IS has as its components the Information Base, a Conceptual Schema, and an Information Processor. The Information Base holds real world information deacriptiona, the Conceptual Schema structurea the Information Base and holda consiatency rulea, and the Information Processor is the uaer interface which allows access and updatea to the Information Baae according to rules in the Conceptual Schema. Since [Chen 76] had presented the E-R model that best deacribed the real world, the structuring of the information deacriptions in the Information Base is based on the E-R model. These extenaions of the E-R model made the schema description easier and increased the expression power of the language, allowed domains to be described hierarchically, and were useful in describing integrity rules. F1 proved to be a tool that provided a practical means of creating information system prototypes.

2.5.2 APPLICATIONS OF THE ENTITY-RELATIONSHIP MODEL

The E-R model's ability to apply real-world views of database design has lead to many diverse applications. [Lee 85] presented various examples of how the ERD techniques could be applied to pictorial database design. The results of this study provides for future applications in knowledge engineering, pattern recognition, Artificial Intelligence, fuzzy language theory, computer graphics, Expert Systems, and pictorial database design.

CRITIAS introduced by [Qian 85] is a Pascal like database programming language data definition facility that provides syntactic structuring of a semantic data model. Based on a formalism of the E-R model [Chen 76], this tool provides for:

1. Data definition, querying, and data manipulation,
2. A consistent means of modeling entities, reference relationships, associations, and subtypes,
3. Integrity constraints are provided at all levels of data abstraction.

This application provided for numerous improvements of the E-R model in the implicit semantics of "ISA" relationships, and improvementa of the textual deacriptiona of the schema.

[Hau 85] developed a two atage E-R model that provided three featurea. The model ia aeparated into a Semantic E-R (SER) uaed to define aemantic entitiea and relationahipa, and the Operational E-R (OER) uaed in mapping to an operational databaae. Thia aeparation allowa each part of the two atage model to reach ita own high level of modeling technique. Secondly, the OER ia normalized, facilitating eaaier mapping into a phyaical databaae with high levela of integrity. Finally, much time and energy is aaved by the databaae deaigner aince the model incorporatea determiniatic algorithma. Applicationa for thia new E-R model have been uaed in computer-integrated manufacturing and varioua other data-driven ayatema.

Uaing the E-R Approach, [Ferrera 85] preaented EASYER, a ayatem for deaigning and documenting databaae applicationa. EASYER, while running on a peraonal computer, aupporta the databaae deaigner in the initial atagea of databaae deaign. The ayatem atorea the documentation of the databaae in ita own organized databaae giving the deaigner a uaeful tool to keep the documentation in line with the operating aoftware of the databaae. By uaing peraonal computera, wideapread uae of the ayatem in a variety of different environmenta becomea a reality.

The E-R model waa uaed to create knowledge baaea (KB) in [Sernadaa 85]. The reaulta of thia application lead to explicit structuring of the KBs allowing for ease of access of inference driven systems. This ties together the previous looseness of casual inference systems with the high structured organizational principles of the E-R model.

Based on the user view in terms of the E-R model [Chen 76], a user-friendly interface for a DBMS was developed [Elmasri 85]. Using the GORDAS [Elmasri 83] query language, a hierarchical view of the database is graphically presented to the user. Light pen sensitive, the graphics can be accessed by the user to form queries that the system converts into a natural language version of the query. This query formalization technique allows a distinction between select conditions and displaying attributes.

2.5.3 ARTIFICIAL INTELLIGENCE AND THE ENTITY-RELATIONSHIP MODEL

With the growing interest in the AI field, many diverse users of this technology have been presented with database design. Knowledge based systems, knowledge engineering techniques, and other AI disciplines have all been reviewed in connection with the E-R model. By combining ERD techniques and KPSP, a knowledge programming system, [Han 85] described how knowledge bases could be designed that better reflect the domain they represent and produced a personnel question-answering system.

[Hawryszkiewycz 85] proposed a system that assists database designers using the E-R model, by creating E-R model sentences that create a dialog with the designer. Creating a database design by taking user inputs, the system evaluates the design by using its knowledge about modeling. This database design creation model uses a heuristic set of rules to show the designer how to create a model that better represents the environment in which it exists.

In order to automate database design using the E-R model, [Staley 85] presented the Object-Relationship Situation (ORS) model. This logical design tool collects information on objects and relationships and stores them in a separate database. This database represents the conceptual schema, giving ORS the capability of accessing the machine readable definitions, based on the PEARL AI programming language. This model uses an extension of E-R modeling specifications as constructs of this logical database representation.

[Briand 85] demonstrated how semantic networks from AI could be used to represent ERDs. Conversion of ERDs into databases was accomplished by using an inference engine (PROLOG) to process ERDs represented by semantic networks. An example was presented of using this translation to create a relational schema.

Database design is a complicated process encompassing numerous steps, each of which include many difficult decision points. Assistance obtained by the database designer with the advent of database design tools insured that effective DBMS were produced. These tools supplemented the design process through manual techniques or computer aided techniques, either of which were a relief to the designer. The Entity-Relationship Approach to database design provided many of the assets these tools had but added a simple and natural interface to the user by using diagrammatic techniques.

This approach to database design has been the focus of much interest and research. The E-R Diagramming technique has been applied to pictorial database design, the E-R Approach has designed and documented database applications, and it has been used to create an interface for a developed DBMS. Artificial Intelligence has been merged with the E-R Approach to: create knowledge bases, represent Entity-Relationship Diagrams, and automate database design. This thesis will define an enhancement to the Entity-Relationship Approach to database design, by incorporating Expert System technology provided by Artificial Intelligence.

Chapter 3

METHOD OF APPROACH

The Entity-Relationship Approach to datsbase design [Chen 76] has been the object of much interest within the DBMS environment. The E-R Approach is a tool which allows the designer to create a database that accurately reflects the enterprise or the users' view of the data. It captures and preserves important semantic information of the real world, and the Entity-Relationship Diagrams allow it to be more easily understood by the database designer. The literature review discussed various theories that enhance this rich database design formalism, but any attempt to encompass them within this thesis would simply impede further enhancement attempts. The E-R model that best suits further modification can be found in [Chen 85], where most of the theoretical difficulties that plsgued the original model were overcome. Included were new rules for translating an E-R logical dstabase from ERDs to data-structures, and new entity types were created that allowed previously troublesome diagramming protocols to be given attribute definitions.

3.1 THE PROBLEM

Even though the E-R model is conceptually simple to understand, an application of the Entity-Relationship Approach to database design can prove to be cumbersome. At any point in the database design process a designer is faced with many difficult decisions, any one of which compounds the possibility of creating an acceptable database. The improvements to date make the model conceptually simpler but do not necessarily reduce the difficulty of making the correct decision.

Many modifications of the E-R model have included methods for translating ERDs into various physical database architectures whether it be relational, hierarchical, or network. A further enhancement to these modifications should be to implement each architecture with an automatic generation of a Data Dictionary. The Data Dictionary provides and manages a database about databases and related categories. It can be used by the database designer to generate reports for unique information needs and locate data redundancies and inconsistencies that can occur over the lifetime of a database.

3.2 THE SOLUTION

The E-R model must be supplemented by a mechanism that will enable a designer to eradicate decision difficulties and provide for automatic generation of a Data Dictionary. This would free the designer from being concerned about the possible mistakes errant decisions would cause and concentrate on designing a DBMS that fully reflects the particular enterprise. The creation of a Data Dictionary would provide the designer a powerful tool for examining the final database to maintain its' integrity.

This thesis describes the design of an Expert Assistant that will monitor the progress of a database designer using the E-R Approach, insuring that the best possible database and Data Dictionary are obtained. Previous formalisms dealing with this area of E-R improvement were only available to the designer if and when the designer requested assistance. This Expert Assistant, in order to be a viable assistance tool, must have the following capabilities:

Make decisions based on the rules defined by [Chen 85].

An explanation facility, including certainty factors, that assists the designer in understanding why a decision should be made in a particular fashion,

Allow for the access of a knowledge base containing the expertise of previous users of the E-R model.

The tool best suited to create an Expert Assistant design that would meet these requirements is the EMYCIN skeletal knowledge engineering system [Waterman 86]. This system uses a rule-based knowledge representation scheme with a rigid backward-chaining control mechanism. The system provides sophisticated explanation and knowledge base acquisition facilities that clearly speed Expert System development.

The Expert Assistant would provide the necessary assistance needed by a designer using the E-R Approach in various ways. Initially, design constructs defined in the E-R model could be controlled to insure that the final product will not become too complex. Placing of attributes and values on entity types can be controlled to insure that the minimal set of definitiona are used. Translating E-R diagrams into data-structure diagrams to be used in varioua DBMS architectures can be scrutinized to maximize the probability that the final database will be viable. Finally, by uaing the information contained in the final databaae deaign, a Data Dictionary could be created that truly reflects the information in the actual database.

By uaing this formalism, the integrity of the rules could be maintained throughout the logical database design which would inaure that at each critical point the designer makes the optimal deciaion. An ability to give experienced auggeations to the designer, when indecision becomea paramount, would allow for the explanation of the ramificationa of making one decision versus another. Thia experience would be represented in a store house of knowledge which could be tapped whenever a mistake has become apparent to the deaigner, the asaistance tool, or both. These protocols would allow the design processa to continue smoothly from start to finish without allowing unnecesaary concern by the deaigner using the E-R Approach.

After a brief definition of the E-R Approach to database design, it will be shown how thia Expert Assiatant could be a valuable tool for assisting the E-R Approach to database deaign. The type of inference engine deaign necesaary to assist this approach without overcoming the independent thought proceaa of the designer will be presented. The method of knowledge representation needed to properly formulate an environment of assistance will also be discussed. A properly designed Expert System merged with the proven attributes of the E-R Approach will provide a tool which will ultimately define the type of formalism that will mandate further work in this area.

Chapter 4

ENTITY-RELATIONSHIP DEFINITION

This chapter describes the Entity Relationship (ER) Approach to database design in its entirety as described by [Chen 85]. This is a simplistic approach because it appears to people to be very natural to use. Since a DBMS should reflect the world it represents, entities and relationships are the obvious choice to represent it. Logical database design is often limiting and restrictive since the designer is often faced with many decisions including data-structure constraints, consideration of access mechanisms, and efficiency of data manipulation. This approach was formalized to relieve the database designer from these difficult decisions and to make the representation of the enterprise easier to comprehend.

The E-R Approach to database design concentrates on designing the conceptual schema, the intermediate step in database design (Figure 4.1). A database designer using Entity Relationship Diagrams (ERD) must produce the entities, the relationships, and the attributes that truly reflect the enterprise. This view of the data is called the "Enterprise Conceptual Schema" or the "Enterprise Schema." The database designer should keep in mind that the enterprise schema should be a pure representation of the real world and not incorporate physical database limitations such as the needs of a particular application program. Figure 4.1

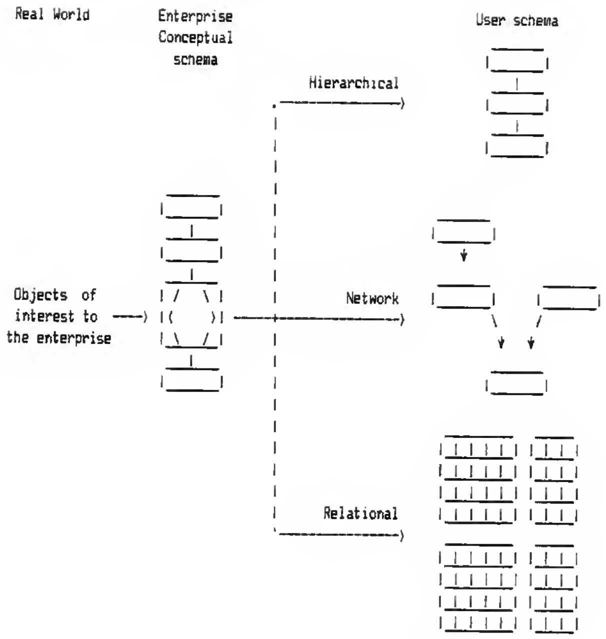

Real World
Enterprise Conceptual schema
User schema
Hierarchical
Objects of interest to the enterprise
Network
Relational

Figure 4.1 Enterprise schema-an intermediate step in logical database design. (Adapted from [Chen 85])

illustrates that the designer must first define the enterprise conceptual schema and then translate it into a user schema for the particular DBMS architecture. This two-phase approach produces the following benefits:

1. Division of functionalities and labor into two phases makes the database design process simpler and better organized.

2. The enterprise schema is easier to design than the final user schema because there are no restrictions levied by the database system (i.e., the enterprise schema is independent of storage and efficiency considerations).

3. The enterprise schema is not as susceptible to change as the user schema. To change from one database system to another, the user schema would probably have to be changed but not the enterprise schema, since the enterprise schema is in principle independent of the database system used. All that would be required is to remap the enterprise schema to a user schema suitable for the new database system. Similarly, if the user schema were changed to optimize a new application program, the enterprise schema would not need to be changed but the enterprise schema would be remapped to a new user schema.

4. The enterprise schema expressed by the entity-relationship diagram is more easily understood by non-EDP personnel. [Chen 85]

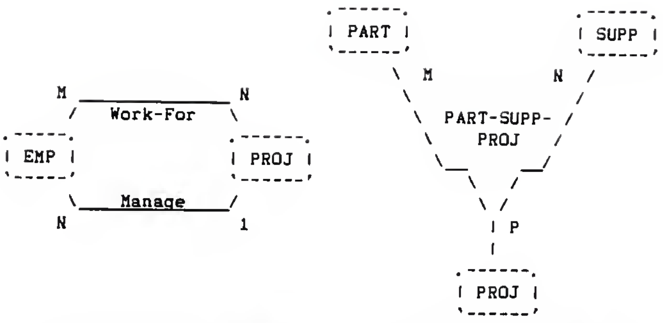
4.1 ENTITY-RELATIONSHIP DIAGRAMS

4.1.1 ELEMENTARY ENTITIES

Elementary entities uniquely identify those objects that are of interest to the enterprise. Represented by rectangular boxes, elementary entities are used to diagram various objects such as EMPLOYEE or STOCK-HOLDER. Not all of the numerous objects found in the real world are of interest to a particular enterprise. Defining entities which are of interest to the enterprise is the responsibility of the database designer.

4.1.2 RELATIONSHIPS

Relationships usually exist between entities and can be classified into different relationship types. Figure 4.2a shows two different



(a) Relationship types are represented by lines.

(b) PART-SUPP-PROJ as a relationship type.

Figure 4.2 [Chen 85]

relationship types, WORK-FOR and MANAGE, between two entity types. A relationship type is denoted by a connecting line between entities. The "N" and "1" notations indicate that one or more projects may have only one manager, and the "M" and "N" notations designate that each project may consist of many employees and each employee may be associated with more than one project.

Relationship types may be used to diagram more than two entity types (Figure 4.2b). PART-SUPP-PROJ is a relationship type for three entity types: PART, SUPP, and PROJ. This three-way relationship (Table 4.1a), can be replaced with three binary relationships: PART-SUPP, SUPP-PROJ, and PROJ-PART as shown in Table 4.1b. If a three-way relationship were constructed from these three binary relationships "nonfacts" are produced (starred

Part #	Supp #	Proj #
68	3	1
68	5	2
10	3	2
10	3	3
17	2	1
17	5	1

(a) Information about PART-SUPP-PROJ relationships

Part #	Supp #
68	3
68	5
17	2
17	5

Supp #	Proj #
3	1
3	2
3	3
5	1
5	2
2	1

Proj #	Part #
1	68
1	17
2	10
2	68
3	10

(b) Information about three binary relationships PART-SUPP, SUPP-PROJ, & PROJ-PART

Part #	Supp #	Proj #
68	3	1
68*	3	2
68*	5	1
68	5	2
10	3	2
10	3	3
17	2	1
17	5	1

(c) Information generated from three binary relationships in Figure 4.1b

Table 4.1 Information concerning three-way relationships. (Adapted from [Chen 85])

entries in Table 4.1c). There are many relationships possible in any given enterprise, the databsse designer must develop only those relationships that are relevant, and must specify the mapping of these relationships (one-to-one, one-to-many, and many-to-many).

4.1.3 COMPOSITE ENTITIES

Composite entity types, disgrammed as specisl rectangular boxes, are entities formed by other entities (i.e., elementary entities or composite entities). To illustrate, SHIPPING is a composite entity type formed by PRODUCT and CUSTOMER (Figure 4.3). Although similar to the relstionship type the composite entity may have properties and a relationship may not. The designing task becomes simpler when the properties sre removed from relationships. Previous work with the E-R Approach sllowed relstionships to have properties. Placing properties on composite entity types and removing them from relationships provides s clesr distinction between entities and relationships.

Composite entities may be built on top of snother composite entities, e.g. HANDLING is formed by the elementary entity type EMPLOYEE and the composite entity type SHIPPING (Figure 4.3). To simplify any confusion in disgramming s composite entity type built on another composite entity type, a special diagramming technique is used. The "component" entity type is connected to the sides of the "composite" entity type (Figure 4.3). Using nouns when naming elementary entity types, verbs for relationship types,

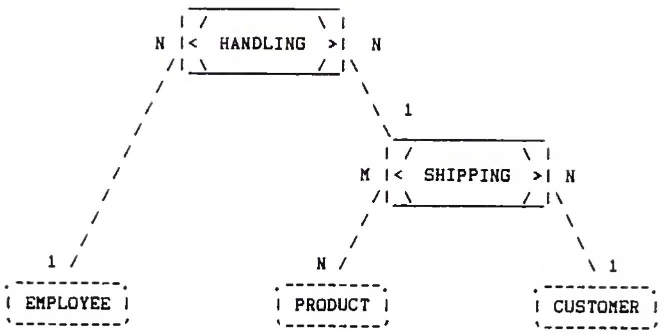


Figure 4.3 A composite entity type can build on top of other composite entity types. [Chen 85]

and gerunds for composite entity types also simplifies the ERD technique.

4.1.4 PROPERTIES OF ENTITIES

The properties of elementary and composite entity types are expressed as attribute-value pairs. In the statement "the AGE of employee X is 24," "AGE" is the attribute of employee X, and "24" is the value of the attribute AGE. The values are categorized into different value types, such as NO-OF-YEARS, QUANTITY, and COLOR. In the ERD technique, value types are represented by circles (in this case parenthesized boxes), and attribute types are represented by arrows that are directed from the entity type to the value type (Figure 4.4). Attributes may have more than one value for any given entity such as the "PHONE-NO" attribute of employee X could

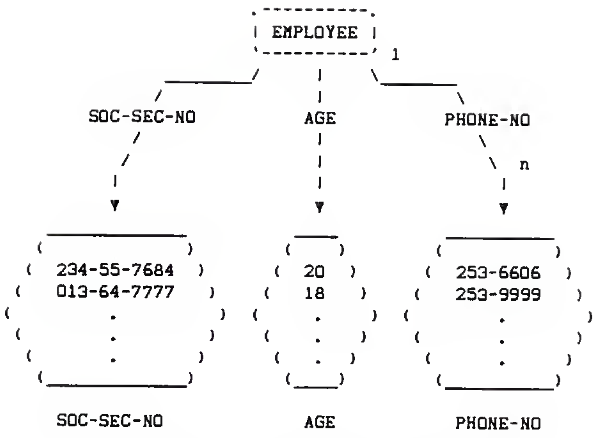


Figure 4.4 Value types and attributes.
(Adapted from [Chen 85])

have more than one value. These multivalued attributes are represented in ERDs by placing a 1:n along the attribute arrow. For simplicity a single valued attribute is not designated with a 1:1 notation.

Figure 4.5 shows how attribute-value pairs are diagrammed with a composite entity. The STARTING-DATE is an attribute of the composite entity WORKING allowing for queries such as "when did employee X start work on project Y". Without the addition of this composite entity this type of query could not be resolved. PERCENTAGE-OF-EFFORT is also an example of another attribute included with the composite entity since it represents the percentage of time that an employee works on a particular project.

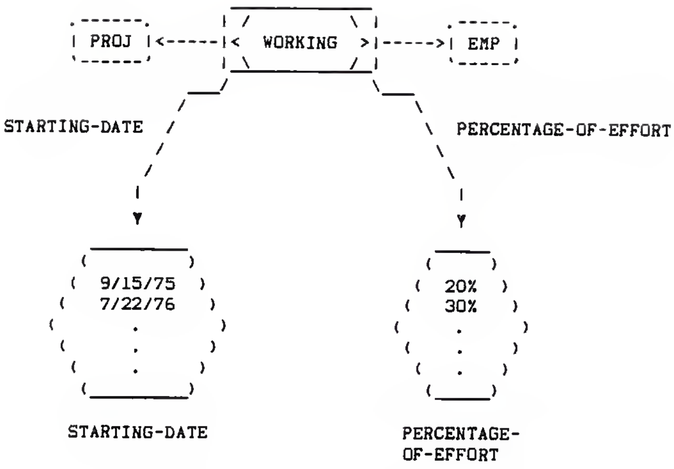


Figure 4.5 Attributes of a composite entity. (Adapted from [Chen 85])

Composite entity attributes are also called the "attribute of relationship" a concept similar to "relationship data" in network database systems and "intersection data" in hierarchical database systems.

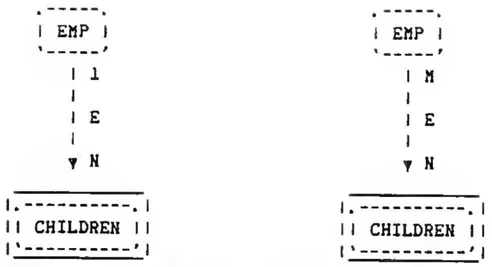
4.1.5 IDENTIFYING ENTITIES AND RELATIONSHIPS

Since entities always have several attributes choosing attribute-value pairs to identify entities must be considered carefully. When placing the attribute that best suits the ERD technique, the designer must pick the attribute that uniquely identifies an entity. If the enterprise is a small business, the attribute NAME would best identify the employees but not if the business is a large enterpriae. In some casea the attributes available are not aufficiently deacriptive and one may have to be created to uniquely identify the entity. Social security numbera, employee numbers, part numbers, and project numbera are aome examplea of creating unique entity identifiera. Thia "entity identifiers" concept is aimilar to the "primary key" in conventional data proceaaing.

Entity identifiers involved in a relation can be used to uniquely identify their relationahips. If PROJ-NO is uaed to identify a project and EMP-NO is used to identify an employee, then the WORK-FOR relationship is identified by both PROJ-NO and EMP-NO. Relationahips may be identified by occurrences of the aame entitiea, e.g. IS-MARRIED-TO ia a relationahip type defined between different occurrences of the entity type PERSON. These relationshipa are not only identified by the entity identifiers but alao by the role the entity plays in the relationahip. The relationahip MARRIAGE can have role names such as HUSBAND and WIFE attached to the entity identifier NAME, where the attached namea are the "rolea" they play in the relationahip.

4.1.6 SPECIAL ENTITIES AND RELATIONSHIPS

Special entity and relationahip typea can be encountered when using the ERD technique. An entity may depend on the exiatence of another entity. A CHILDREN entity exista only if aasociated employeea exiat in the databaae. If an employee involved in this relationahip leavea the company there may be no need to keep a



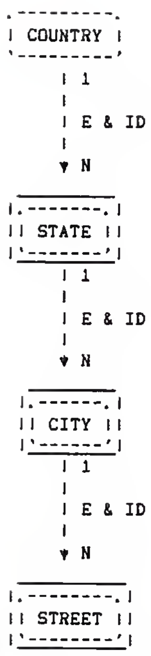
(a) An existence dependent relationship type and a weak entity type.

(b) An existence dependent relationship may also be a many-to-many mapping.

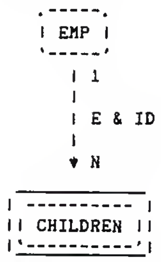
Figure 4.6 (Adapted from [Chen 85])

record of the CHILDREN entity. The diagram that represents this "weak" entity type is a double-rectangular box (Figure 4.6a). The "E" in the figure indicates that the relationship is "existence-dependent", and the arrow indicates the direction of dependency. "Existence-dependent" relationships may be possible in a many-to-many mapping, for instance if a father leaves a company the CHILDREN entity may still exist if the mother is an employee of the company (Figure 4.6b).

When an entity is identified by using relationships with other entities, an "ID dependency" on the other entities is developed. In order to uniquely identify a street address the city, state, and country must be known (Figure 4.7a). This dependency is indicated by placing an "ID" along the relationship line and, as in "existence dependencies", the arrow indicates the direction of the



(a) Existence dependency and ID dependency



(b) Existence dependency and ID dependency

Figure 4.7 (Adapted from [Chen 85])

dependency. Most "ID dependencies" are associated with "existence dependencies", but "existence dependencies" do not always imply an "ID dependency". The CHILDREN entity in Figure 4.7b is identified with its own attribute(s) and the parent(s)' ID (Table 4.2a), while the CHILDREN entity in Figure 4.6a may be identified by its own CHILDREN-NO (Table 4.2b).

4.2 TRANSLATING E-R DIAGRAMS INTO DATA-STRUCTURE DIAGRAMS

Logical data structures of network database systems can be expressed in terms of data-structure diagrams as in Figure 4.8. The rectangular boxes represent a record type such as EMP or DEPENDENT, the arrow represents a data-structure set which connects two record types. The owner record type of the data-structure set is located at the originating end of the arrow, and the member record type of the data-structure set is located at the end of the arrow. The owner record may have zero, one, or more member records and a member record may have only one owner record. Figure 4.9 shows that each EMP (employee) record may be connected to many DEP (dependent) records or to none, but each DEP record must be associated with exactly one EMP record. A one-to-many (1:N) association between the owner record type and the member record type can be represented by an arrow or be illustrated in a table format as in Table 4.3.

Figure 4.10 depicts an EMP record type as the owner record type in a data-structure set with EMP-SKILL (employee skill) as the member

CHILDREN ID		Data about CHILDREN	
	EMP ID		
Name	Parent's SSN	Age	Medical Insurance
Nancy Bok	013-58-5545	12	BC/BS
Lawrence Bok	172-66-6672	5	BC/BS
Robert Johnson	819-38-7761	21	Has its own policy

(a) ID dependency.

CHILDREN ID			
CHILDREN NO	Name	Age	Medical Insurance
1011	Nancy Bok	12	BC/BS
1025	Lawrence Bok	21	BC/BS
1044	Robert Johnson	5	Has its own policy

(b) No ID dependency.

Table 4.2 (Adapted from [Chen 85])

```
          .-----.
          | EMP |  "Owner" record type
          '-----'
             |
             |
             |  <----- Data-structure-set
             |
             v
      .-----------.
      | DEPENDENT |  "Member" record type
      '-----------'
```

Figure 4.8 A data-structure diagram. [Chen 85]

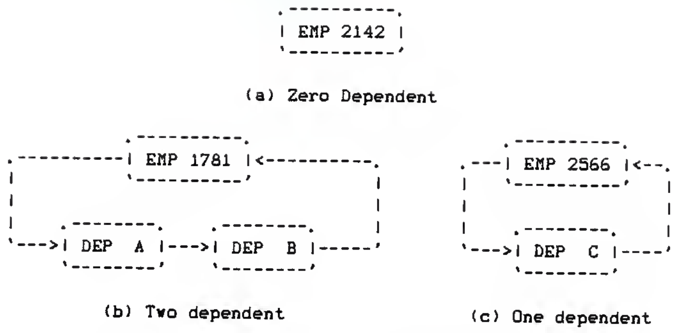


Figure 4.9 An owner record may have zero, one or more member records. (Adapted from [Chen 85])

EMP-NO	DEPENDENT
1781	A
1781	B
1781	C
2566	D
.	.
.	.
.	.

Table 4.3 One-to-many correspondence between EMPLOYEE and DEPENDENTS. [Chen 85]

record type. EMP-SKILL is also a member record type of another data-structure set whose owner record is the record type SKILL. The EMP-SKILL record type contains cross-reference information

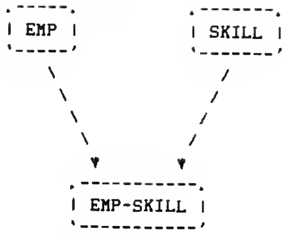


Figure 4.10 Two data-structure sets have the same member record type. [Chen 85]

concerning the EMP and SKILL record types and can be represented in tabular format as seen in Table 4.4.

Table 4.4 also shows that an employee may have more than one skill and that a particular skill may be known by more than one employee. This information creates a many-to-many (M:N) association between employees and skills and can be derived from the data-structures in Figure 4.10 and the cross-reference information in Table 4.4. A 1:M mapping exists between the EMP record type and the EMP-SKILL record type, and similarly the mapping between the SKILL record type and the EMP-SKILL is 1:N. This information shows that the correspondence between the EMP record type and the SKILL record type is a M:N mapping.

Implementing the data-structure diagram in Figure 4.10 can be accomplished by using pointer arrays as shown in Figure 4.11. The data-structure set between the EMP record type and the EMP-SKILL record type is represented with dashed-lines, and the data-

EMP-NO	SKILL
2142	COBOL
2142	PL/1
1781	COBOL
2566	PL/1
.	.
.	.
.	.

Table 4.4 Cross-reference information about EMPLOYEES and SKILLS. [Chen 85]

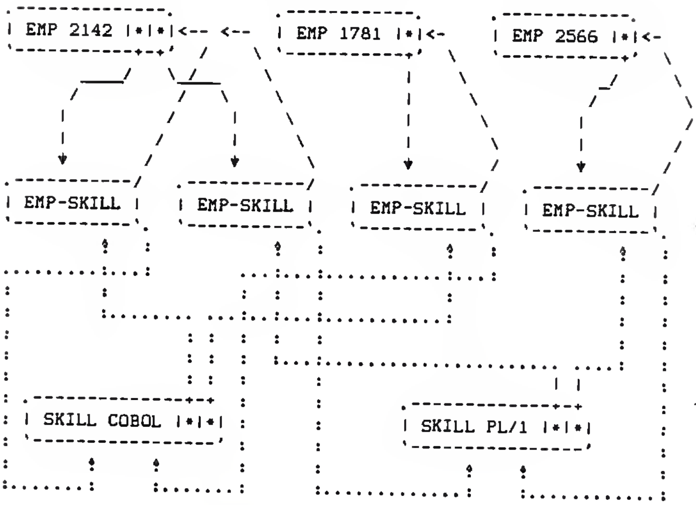


Figure 4.11 Implementation of the data-structure sets in Figure 4.10 as pointer arrays. (Adapted from [Chen 85])

structure set between the SKILL record type and the EMP-SKILL record type is represented with dotted-lines.

Determining the skill of a particular employee (e.g., 2142) can be accomplished in a few steps. First, locate the EMP record type with the particular EMP-NO of interest. By using the dashed lines, the first EMP-SKILL record type related to this employee is found. Next the SKILL record with a skill-name equal to COBOL is located by using the dotted-line pointers. The dashed line is used again to locate the second EMP-SKILL record related to the same employee. The dotted-line pointer is then used to find the SKILL record with skill-name equal to PL/1. Finally, since no more EMP-SKILL records can be located that correspond to the EMP record of interest, the skills of the employee with EMP-NO = 2142 are determined to be COBOL and PL/1.

All the employees with a particular skill can also be found using pointer arrays. This search is started by locating the SKILL record type with the SKILL-NAME equal to the skill of interest, e.g., COBOL. By using the dotted-lines all the EMP-SKILL record types are retrieved that are related to this SKILL record. From these EMP-SKILL records the EMP records can be found by following the dashed-lines. These EMP records represent those employees that have the SKILL equal to COBOL and have EMP-NOs equal to 2142 and 1781.

"Chains" (Figure 4.12) may also be used to implement the data-

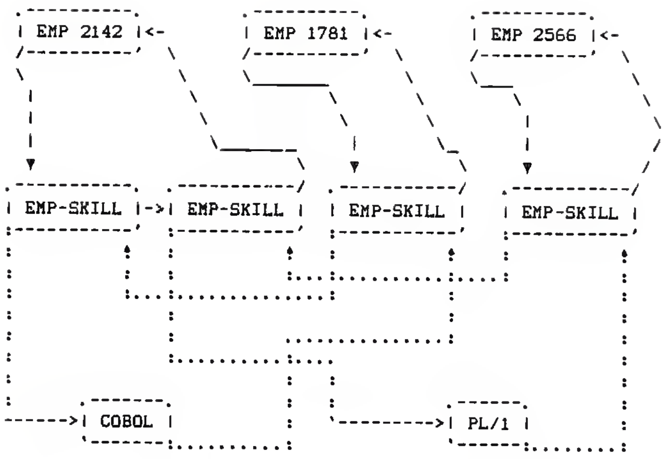


Figure 4.12 Implementation of the data-structure sets in Figure 4.10 as chains. (Adapted from [Chen 85])

structure diagram in Figure 4.10. The dashed-line chains connect all the EMP records with the EMP-SKILL records, and the dotted-line chains connect all the SKILL records with the EMP-SKILL records. To find the skills of a particular employee, say the EMP record with a EMP-NO equal to 2142, the first EMP-SKILL record must be found for this EMP by following the dashed-line chain. Then by following the dotted-line chain, the corresponding SKILL record can be located. The next EMP-SKILL record can be found by using dashed-line chain and its' corresponding SKILL record can be located as before. Finally, since there does not exist any further EMP-SKILL records (via the dashed-line chain), all the information

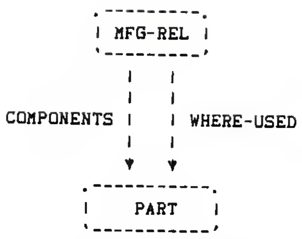


Figure 4.13 Two data-structure sets have the same owner and member record types. [Chen 85]

about the skills of EMP with the EMP-NO equal to 2142 have been found. As with the pointer array implementation, each employee with a particular skill may also be found.

Another type of data-structure diagram is shown in Figure 4.13. PART and MFG-REL (manufacturing-relationship) are the two record types, where each product to be manufactured consists of many components or parts, and each part is in turn made up of other parts. The PART record type contains information about the particular part. The MFG-REL record type contains information about the relationship between parts. Table 4.5 shows that each PART #1 is composed of five PART #2's and two PART #3's, and that each PART #3 is used as a subpart of PART #1 and PART #4. The two data-structure sets in Figure 4.13 can be implemented as chains (seen in Figure 4.14) where the dashed-lines represent the COMPONENT chain, and the dotted-lines represent the WHERE-USED chain.

SUPER-PART-NO	SUB-PART-NO	QTY
1	2	5
1	3	2
4	3	1
4	5	2

Table 4.5 Manufacturing relationship between parts. [Chen 85]

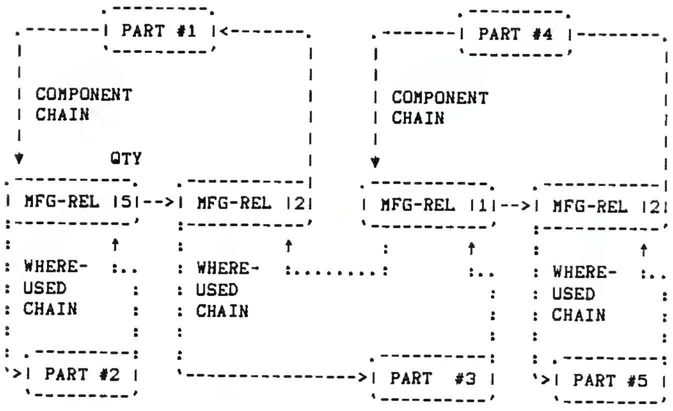


Figure 4.14 Implementation of the data-structure sets in Figure 4.13. (Adapted from [Chen 85])

To discover the components of a particular part, the MFG-REL records are retrieved by using the COMPONENT chain and then by retrieving all the subparts using the WHERE-USED chain. This reveals that PART #4 consists of one PART #3 and two PART #5's. To find where a particular PART is used, the MFG-REL records related to the PART are retrieved by using the WHERE-USED chain, and the corresponding PART records are retrieved by using the COMPONENT chain. This shows that two PART #5's are used in manufacturing a PART #4. Figure 4.8, 4.10, and 4.13 are the basic representations of data-structure set diagrams. Any database can be expressed in a large data-structure set diagram based on these three building blocks.

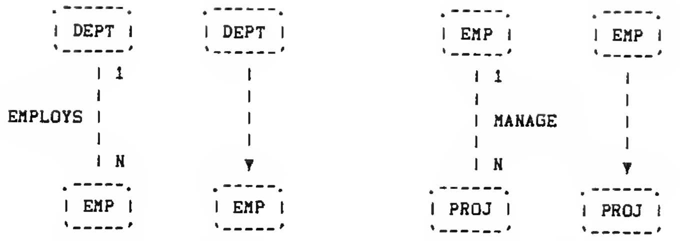
4.2.1 TRANSLATION RULES

Data-structure diagrams are closer to the actual physical organization of the database than the Entity-Relationship diagrams. It is recommended that the database designer first diagram the enterprise view of the data using E-R diagrams and then translate them into data-structure diagrams. This is much simpler than developing the data-structure diagrams directly from the data of interest within the enterprise. Several rules are necessary to make this conversion from E-R diagrams to data-structure diagrams. The following translation rules are based on the relationships between entities:

1. <u>Relationships defined on two different entity types:</u>

   a. A 1:1 or 1:N relationship.

      The DEPT-EMP relationship type in Figure 4.15a is a 1:N mapping and can be transformed into the data-structure diagram in Figure 4.15b. The entity types DEPT and EMP in the E-R diagram are treated as record types in the data-structure diagram,



(a) ERD (b) Data-structure diagram (c) ERD (d) Data-structure diagram

Figure 4.15 (Adapted from [Chen 85])

while the relationship type EMPLOYS is represented by a data-structure set (an arrow) in the data-structure diagram. Similarly, the 1:N relationship type EMP-PROJ in Figure 4.15c can be transformed into the data-structure diagram in Figure 4.15d. The entity types EMP and PROJ in the E-R diagram are treated as record types in the data-structure diagram, while the relationship type MANAGE is represented by a data-structure set.

b. A M:N relationship.

The relationship type WORK-FOR in Figure 4.16a is a M:N mapping and is translated into the data-structure diagram shown in Figure 4.16b. A relationship type with a M:N mapping will be translated into a record type with two arrows pointing from the related entity record types. Therefore the relationship type WORK-FOR was not translated into a data-structure set, but into a record type. The PROJ-EMP record type is usually called a "relationship record type" or a "dummy record type." Similarly, since the relationship type HAVE in Figure 4.16c is a M:N mapping, it is also translated into the "relationship record type" EMP-SKILL in the data-structure diagram.

2. <u>Relationships defined on three or more entity types:</u> The relationship type in an E-R diagram is translated into a relationship record type in the data-structure diagram no matter whether the relationship is a 1:1,

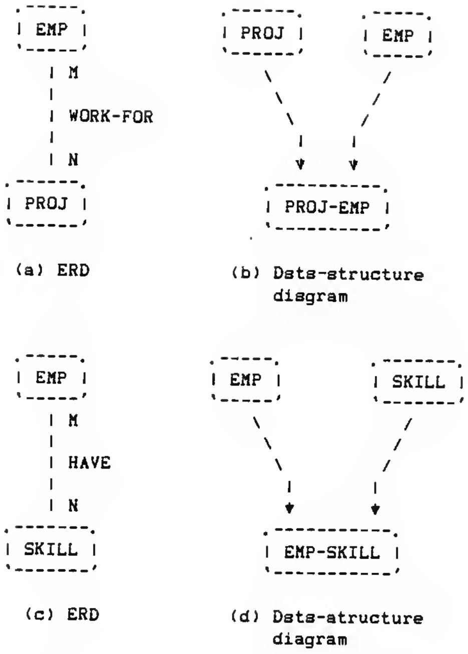
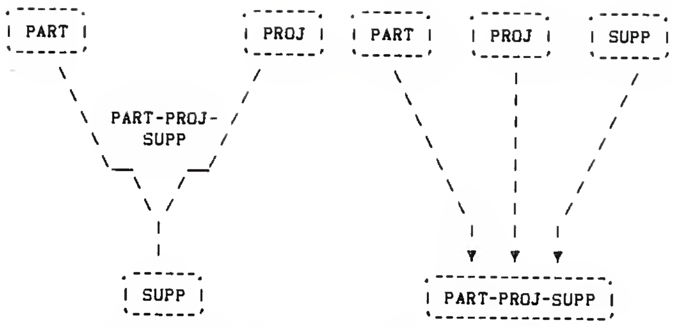


Figure 4.16 (Adapted from [Chen 85])

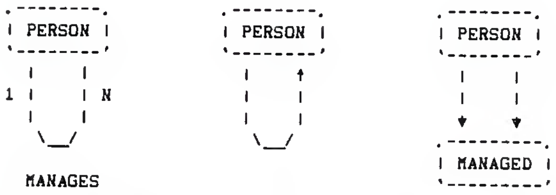
1:N, or M:N. The PART-PROJ-SUPP relationship type in Figure 4.17a is a relationship type defined by three entity types and will be translated into a record type in the data-structure diagram shown in Figure 4.17b.

3. <u>Binary relationships defined on the same entity types:</u> If a binary relationship is a 1:N association, a relationship type such as MANAGES in Figure 4.18a can be transformed into at least two possible data-structure diagrams (Figures 4.18b and c). Most network database systems do not allow the same record type to be used as both the owner record type and the member record type of a data-structure set. This makes the data-structure diagram in Figure 4.18b illegal. The data-structure diagram in Figure 4.18c can be used as an example of the data-structure



(a) E-R diagram (b) Data-structure diagram

Figure 4.17 [Chen 85]



(a) E-R diagram (b) (c) Data-structure diagram

Figure 4.18 (Adapted from [Chen 85])

counterpart of the E-R diagram in Figure 4.18a. For binary relationships with other types of mapping, the same type of data-structure diagram is used, like the M:N relationship type CONSISTS-OF (MFG-REL) seen in Figure 4.19a and its equivalent data-structure diagram shown in Figure 4.19b.

4. Composite entity types:
All composite entity types are translated into record types. The composite entity types SHIPPING and HANDLING in Figure 4.20a are translated into the

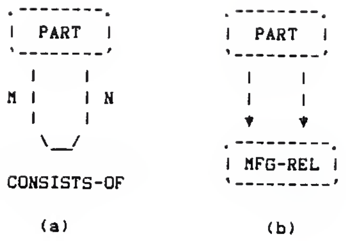


Figure 4.19 [Chen 85]

record types SHIPPING and HANDLING in Figure 4.20b. The "component entity types" become the "owners" of the data-structure sets, and the composite entity types become the "members" of the data-structure sets. For example, PRODUCT and CUSTOMER are owners of the data-structure sets in which SHIPPING is the member. Similarly, EMP and SHIPPING are the owners of the data-structure sets in which HANDLING is the member (Figure 4.20b).

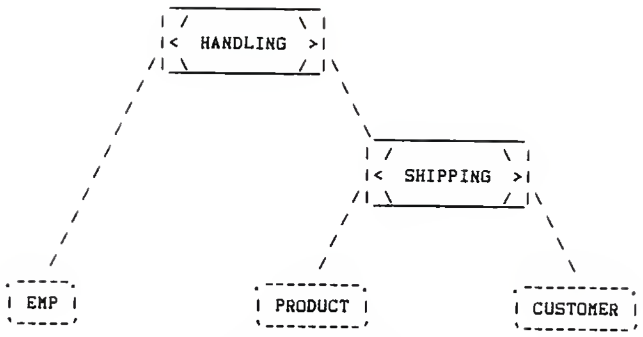
4.3 LOGICAL DATABASE DESIGN STEPS

[Chen 85] formalized a description of the major steps involved in logical database design used by the E-R Approach:

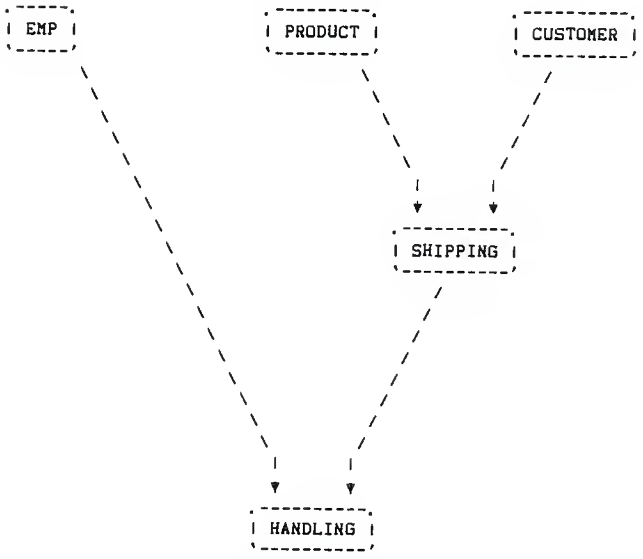
1. Draw an initial E-R diagram.
   a. Identify elementary entity types.
   b. Identify relationships between elementary entity types.

2. Refine the E-R diagram.
   a. Convert some relationship types into composite entity types.
   b. Identify "new" relationship types and high-level composite entity types.
   c. Repeat subsets a and b until no more new relationship types and composite entity types can be found.

3. Draw an attribute diagram for entity types.



(a) E-R diagram



(b) Data-structure diagram

Figure 4.20 [Chen 85]

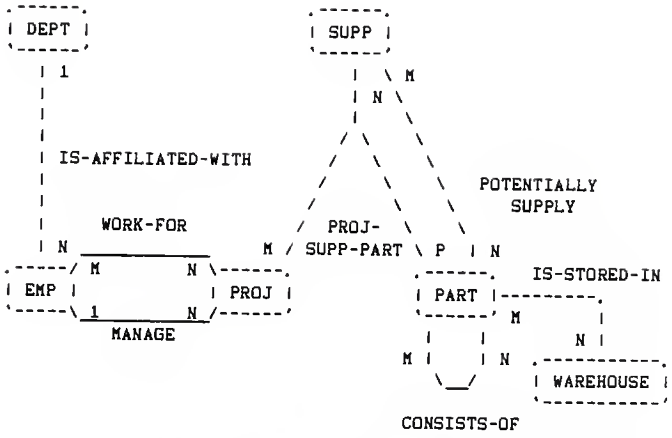
4. Convert the E-R diagram into one of the following:
   a. A data-structure diagram for CODASYL DBMS's.
   b. A hierarchical diagram for hierarchical DBMS's.
   c. A set of relations (tables) for relational DBMS's.

4.3.1 AN INITIAL E-R DIAGRAM

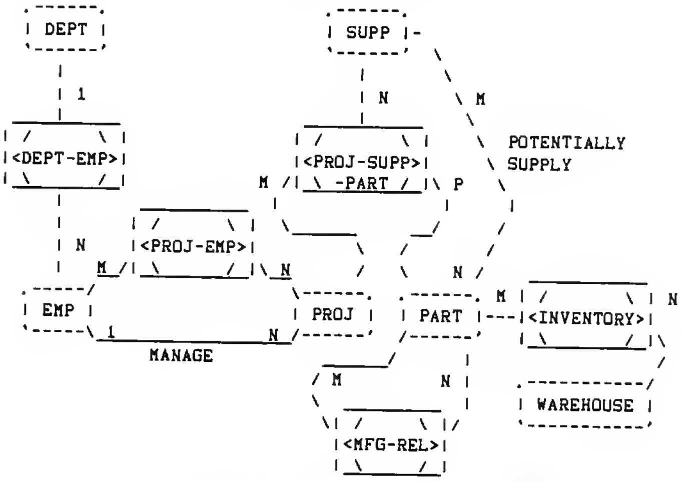
A simple manufacturing company is used as an example of the enterprise of interest to use with the Entity Relationship Approach to logical database design. The elementary entity types of interest in this enterprise are identified as: EMP, PROJ, DEPT, PART, and SUPP (Figure 4.21a). Identifying relationship types between elementary entity types (Step 1b), begins with defining the relationship types on only one entity type, then on two entity types, and then on three or more entity types. In this example the following relationship types are defined:

1. <u>Relationship types defined on one entity type</u>:
   a. The CONSISTS-OF relationship type describes the superparts and subparts of a given part. This is the only relationship type of interest in this category.

2. <u>Relationship types defined on two entity types</u>:
   a. The IS-AFFILIATED-WITH relationship type describes the employees affiliated with a given department and is a 1:N mapping.
   b. The WORK-FOR relationship type describes the project affiliations of all the employees and is a M:N mapping. That is, an employee can work for many projects, and a project can involve many employees.
   c. The MANAGE relationship type identifies the managers of projects and is a 1:N mapping. That is, a project has at most one manager, but an employee can manage several projects.
   d. The POTENTIALLY-SUPPLY relationship describes the list of potential suppliers for a given part and is a M:N mapping.
   e. The IS-STORED-IN relationship type describes which



(a) An initial E-R diagram for a manufacturing company



(b) An E-R diagram for a manufacturing company

Figure 4.21 (Adapted from [Chen 85])

part is stored in which warehouse and is a M:N mapping.

3. <u>Relationship types defined on three or more entity types</u>:
   a. The PROJ-SUPP-PART relationship type describing which supplier supplies which part for a particular project is a many-to-many-to-many (three-way) relationship. That is, for a given part, there may be many suppliers who can supply this part to many projects. Likewise, any project may use many parts from different suppliers.

4.3.2 REFINE THE E-R DIAGRAM

Step 2a requires that each relationship type be examined to see if there is a need to record relevant data concerning it. If this is warranted, the relationship will be converted into a composite entity type. Figure 4.21a shows the relationship types concerning the manufacturing company and the following five conversions to composite entities are done:

1. The IS-AFFILIATED-WITH relationship type is converted into the DEPT-EMP composite entity type.
2. The WORK-FOR relationship type is converted into the PROJ-EMP composite entity type.
3. The PROJ-SUPP-PART relationship type is converted to a composite entity type with the same name.
4. The CONSIST-OF relationship type is converted into the MFG-REL composite entity type.
5. The IS-STORED-IN relationship type is converted into the INVENTORY composite entity type.

Step 2b is not necessary since there are no other relationship types or high-level composite entity types of interest. Figure 4.21b shows the results of applying Step 2 to the manufacturing company.

4.3.3 ATTRIBUTE DIAGRAM FOR ENTITY TYPES

Attribute diagrams for established entity types are created in the third step of logical database design. Figure 4.22 shows the attributes and value types for the DEPT and EMP entity types, and the DEPT-EMP composite entity type. The entity types are shown in the upper conceptual domain and the attribute and value types are in the lower conceptual domain. DEPT has three attributes: DEPT-NO, THIS-YEAR-BUDGET, and LAST-YEAR-BUDGET. EMP has five attributes: EMP-NO, BIRTH-DATE, SALARY, HOME-PHONE, and OFFICE-PHONE. Attributes might not have the same names as the value

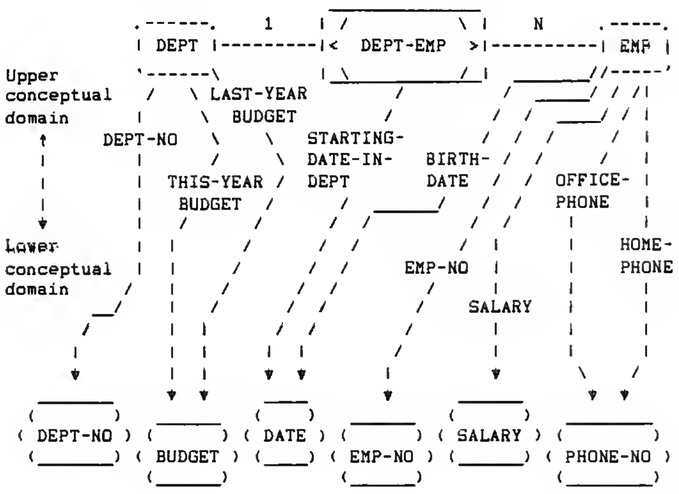


Figure 4.22 Attributes and value types for DEPT, EMP, and DEPT-EMP. (Adapted from [Chen 85])

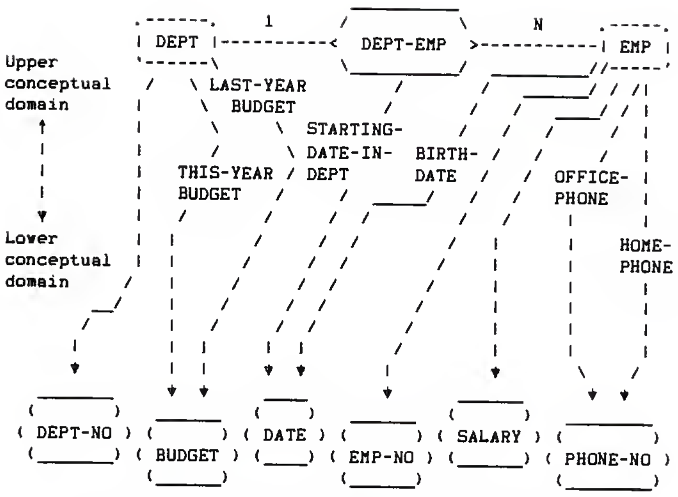


Figure 4.23 A simplified version of Figure 4.22. (Adapted from [Chen 85])

types, and it is possible to have more than one attribute relating to the same value type. For example, the attributes THIS-YEAR-BUDGET and LAST-YEAR-BUDGET attributes of the entity type DEPT have the same value type BUDGET. In order to simplify the diagram, attribute names are omitted if they have the same name as the value types (Figure 4.23).

Attribute and value types for the PROJ and EMP entity types and the PROJ-EMP composite entity type in Figure 4.21b, are shown in Figure 4.24. There are five value types: %EFFORT, DATE, PROJ-NO, BUDGET, and PROJ-NAME and five attributes types: %EFFORT, STARTING-DATE-IN-PROJ, PROJ-NO, BUDGET, and PROJ-NAME. The attributes of

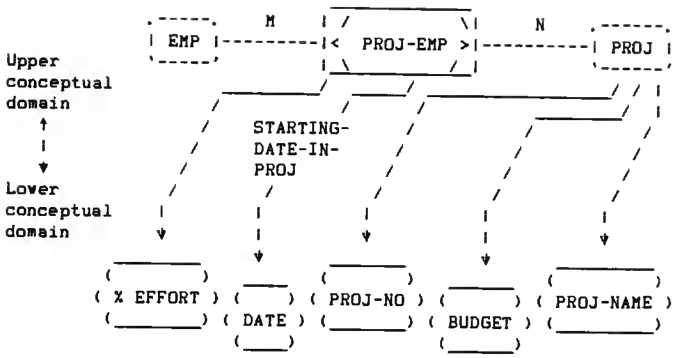


Figure 4.24 Attributes and value types for PROJ and PROJ-EMP. (Adapted from [Chen 85])

the composite entity PROJ-EMP are STARTING-DATE-IN-PROJ (which is the date that the employee started working for a project) and %EFFORT (which is the percentage of time that an employee is expected to spend on a particular project). The attribute and value types for the EMP entity type have been diagrammed in Figure 4.23. Figure 4.25 shows the value and attribute types for the entity types SUPP and PART and the composite entity type PROJ-SUPP-PART. The entity SUPP has the attributes SUPP-NO and ADDRESS, and the entity PART has the attributes PART-NO, WEIGHT, AND COLOR. The composite entity type PROJ-SUPP-PART has the attribute QTY (which is the quantity of a certain part supplied by a certain supplier to a certain project). The attributes of the PROJ entity have already been shown in Figure 4.24.

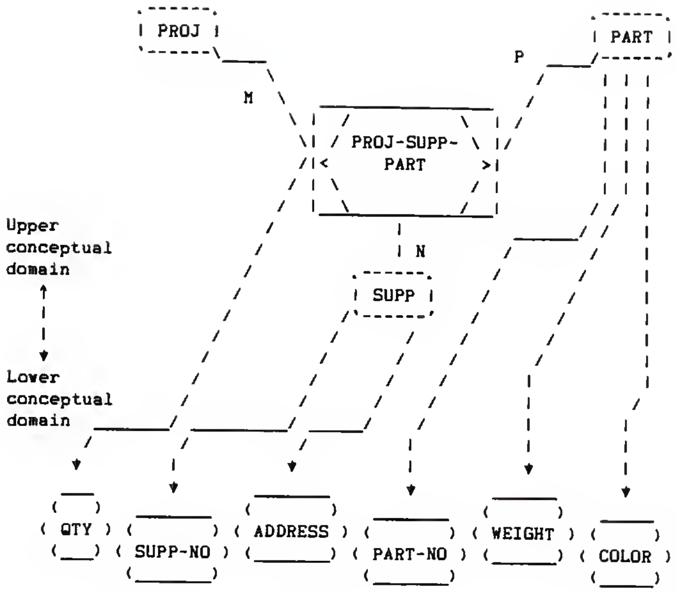


Figure 4.25 Attributes and value types for SUPP, PART, and PROJ-SUPP-PART. (Adapted from [Chen 85])

The attributes and value types for the WAREHOUSE, INVENTORY, and MFG-REL entities are shown in Figure 4.26. The WAREHOUSE elementary entity has WAREHOUSE-NO and ADDRESS as attribute types, and the INVENTORY composite entity has QTY-ON-HAND as an attribute type (which is the quantity of a part stored in a warehouse). The MFG-REL composite entity has an attribute type of QTY-FOR-MFG (representing the quantity of a subpart needed to make a super-part). The QTY-ON-HAND and QTY-FOR-MFG attribute types have the same value type QTY. The Figures (4.23 through 4.26) illustrate

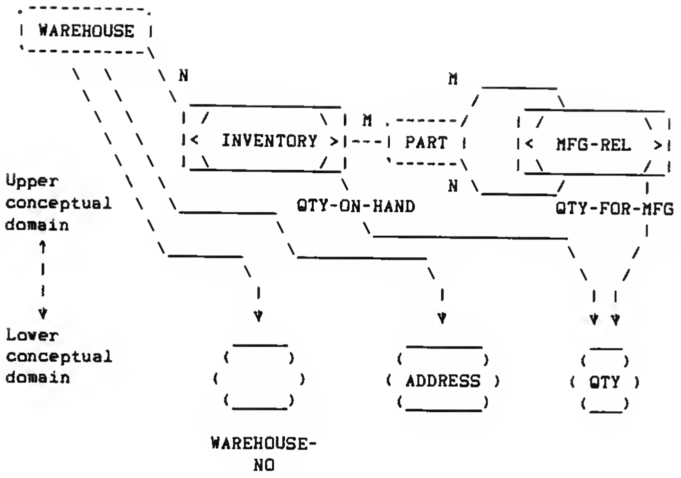


Figure 4.26 Attributes and value types of WAREHOUSE, INVENTORY, and MFG-REL. (Adapted from [Chen 85])

all the attributes and value types necessary to describe the properties of the entities that may be of interest to the manufacturing company.

4.3.4 TRANSLATE THE E-R DIAGRAM

First the E-R diagram in Figure 4.21b is translated into the data-structure diagram shown in Figure 4.27. All elementary and composite entity types become record types in the data-structure diagram. The MANAGE relationship type is a 1:N mapping so it is translated into a data-structure set (i.e., an arrow). Since the relationship type PROJ-EMP is a M:N mapping, it is translated into

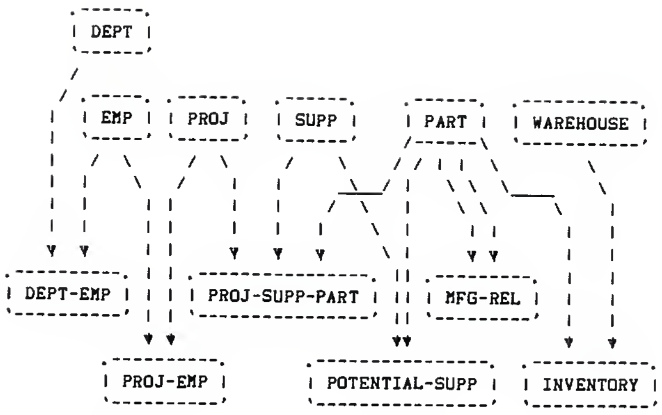


Figure 4.27 The data-structure diagram derived from the E-R diagram in Figure 4.21. (Adapted from [Chen 85])

a record type with arrows from the related entity record types EMP and PROJ.

4.3.5 DESIGN RECORD FORMAT

Deciding how to group the attributes of entities into records and how to implement the data-structure sets ("chains"?, "pointer arrays"?, etc.) is based on the following:

> All the attributes of an elementary or composite entity will be put into the same record type. For example, the attributes of DEPT will be treated as the names of fields in the DEPT record type (see Figures 4.23 and 4.28). (Adapted from [Chen 85])

After placing attributes on the record types, the next step is to decide how to implement the data-structure sets. In this example

```
.-------------------------------------------------------.
| DEPT-NO | THIS-YEAR-BUDGET | LAST-YEAR-BUDGET |*|
'-----------------------------------------------+'
                                                 |
                                                /
                                               /
                                              /
                                             |
                                             v
                                   To the first DEPT-EMP
                                     record related to
                                      this department
```

Figure 4.28 DEPT record. [Chen 85].

"chains" are used as the physical implementation of the data-structure sets. Figures 4.12 and 4.14 will be used as the physical implementation of Figures 4.10 and 4.13, respectively. Allowing for the following observations on how to implement chain pointers:

1. If the record is the owner record type of a data-structure set, it should have a pointer to the first member record occurrence.

2. If the record is a member record type of a data-structure set, it should have a pointer to the next member record occurrence in the chain or, if it is the last record in the chain, to the owner record occurrence.

3. If a record type is involved in multiple data-structure sets, it should contain several pointers, one for each data-structure set. [Chen 85]

These rules define the pointers for the record types and can be seen in Figures 4.28 through 4.34. Since the DEPT record in Figure 4.28 is the owner record type of a data-structure set, it has a pointer to the first DEPT-EMP record occurrence related to this department. Figure 4.29 shows that the EMP record has three

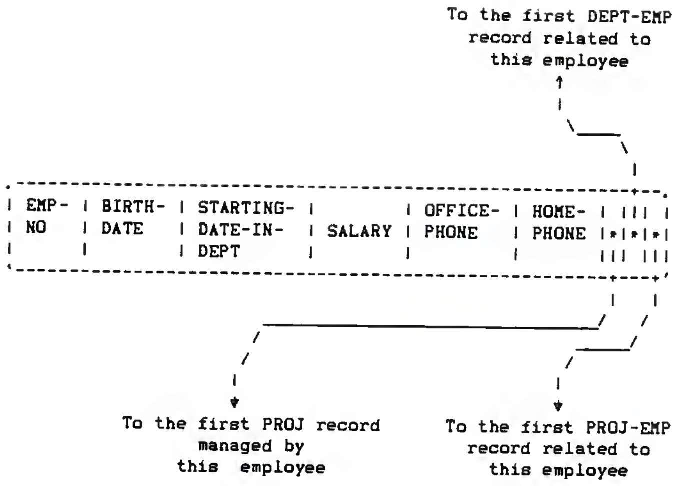
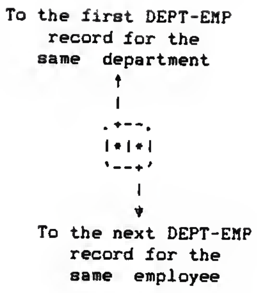

To the first DEPT-EMP record related to this employee
EMP-NO
BIRTH-DATE
STARTING-DATE-IN-DEPT
SALARY
OFFICE-PHONE
HOME-PHONE
To the first PROJ record managed by this employee
To the first PROJ-EMP record related to this employee

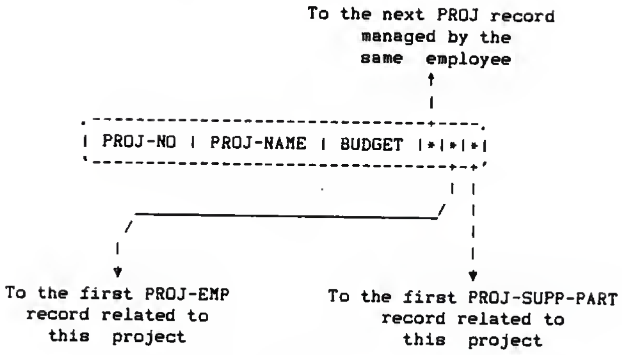
Figure 4.29 EMP record. [Chen 85].

pointers since it is involved in three data-structure sets. Because the EMP record type is the owner record of the data-structure set with member record type PROJ, it keeps a pointer to the first PROJ record occurrence managed by this employee. The value of the pointer is null if the employee is not a manager of any project. Since the EMP record type is also the owner record type of the data-structure set whose member record type is PROJ-EMP, it must also maintain a pointer to the first PROJ-EMP record occurrence in the chain.

The DEPT-EMP record maintains two pointers since it is the member record type of two data-structure sets, the DEPT-EMP record


To the first DEPT-EMP record for the same department
* | *
To the next DEPT-EMP record for the same employee

(a) DEPT-EMP record


To the next PROJ record managed by the same employee
PROJ-NO
PROJ-NAME
BUDGET
* | * | *
To the first PROJ-EMP record related to this project
To the first PROJ-SUPP-PART record related to this project

(b) PROJ record

Figure 4.30 (Adapted from [Chen 85])

```
                                 To the next PROJ-EMP
                                    record for the
                                    same  employee
                                          ↑
                                          |
.-------------------------------------+--.
| STARTING-DATE-IN-PROJ | %EFFORT |*|*|
'-------------------------------------+-'
                                        |
                                        ▼
                                 To the next PROJ-EMP
                                    record for the
                                     same project
```

(b) PROJ-EMP record

```
          To the first PART-SUPP-PROJ
       record related to this supplier
                       ↑
                       |
.-------------------+--.
| SUPP-NO | ADDRESS |*|*|
'---------------------+'
                       |
                       ▼
          To the first POTENTIAL-SUPP
           related to  this supplier
```

(b) SUPP record

Figure 4.31 (Adapted from [Chen 85])

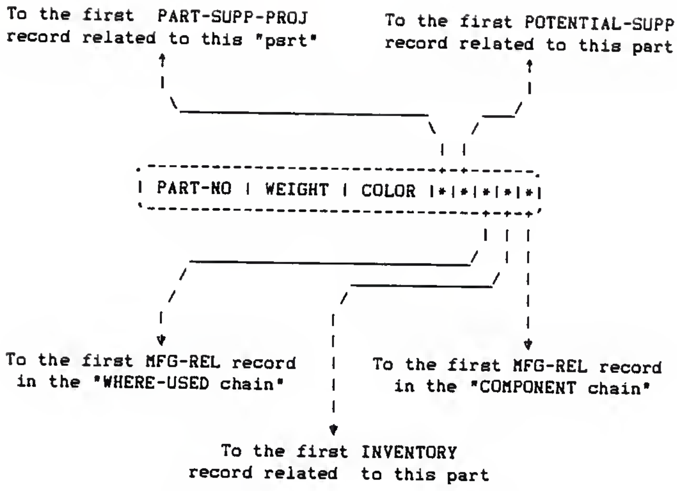
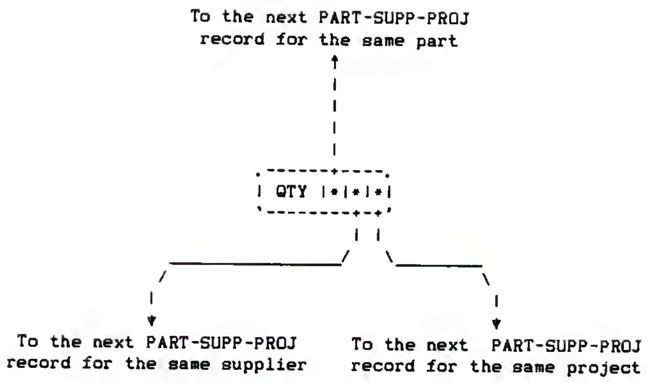

To the first PART-SUPP-PROJ record related to this "part"
To the first POTENTIAL-SUPP record related to this part
PART-NO
WEIGHT
COLOR
To the first MFG-REL record in the "WHERE-USED chain"
To the first MFG-REL record in the "COMPONENT chain"
To the first INVENTORY record related to this part

Figure 4.32 PART record. [Chen 85]

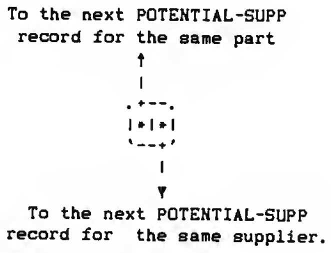
occurrence for the same department, and the DEPT-EMP record occurrence for the same employee (see Figures 4.27 and 4.30a). A more complicated case can be seen with the PROJ-SUPP-PART record type in Figures 4.27 and 4.33a. Since this record type is the member record type of three data-structure sets, it has three pointers, one for each chain. Similar explanations can be given for the pointers in the other record types.

4.4 DESIGN CONSIDERATIONS

The translations rules from E-R diagrams into data-structure diagrams that were discussed in Section 4.2.1 are not the only rules of translation. A simple rule could be used to translate all


To the next PART-SUPP-PROJ record for the same part
QTY
To the next PART-SUPP-PROJ record for the same supplier
To the next PART-SUPP-PROJ record for the same project

(a) PROJ-SUPP-PART record


To the next POTENTIAL-SUPP record for the same part
To the next POTENTIAL-SUPP record for the same supplier.

(b) POTENTIAL-SUPP record

Figure 4.33 (Adapted from [Chen 85])

```
.--------------------------.
| WAREHOUSE-NO | ADDRESS |*|
'-------------------------+'
                          |
                          v
           Too the first INVENTORY
              record related to
               this  warehouse
```

(a) WAREHOUSE record

```
        To the next INVENTORY
           record for the
            aame  part
                 ^
                 |
  .--------------+--.
  | QTY-ON-HAND |*|*|
  '----------------+'
                   |
                   v
          To the  next INVENTORY
             record for the
             aame warehouse
```

(b) INVENTORY record

Figure 4.34 (Adapted from [Chen 85])

relationship types into record types no matter what types of mapping they are. Implementing this rule allows the E-R diagram in Figure 4.21 to be tranalated into the data-structure diagram in Figure 4.35 instead of the diagram shown in Figure 4.27.

With this simplified rule the data-structure diagram will be more complicated and be lesa efficient in retrieval and updating. But it may allow for a higher level of data independence since programs and database structures would not need to be changed when a particular relationship type changea from a 1:N mapping to a M:N

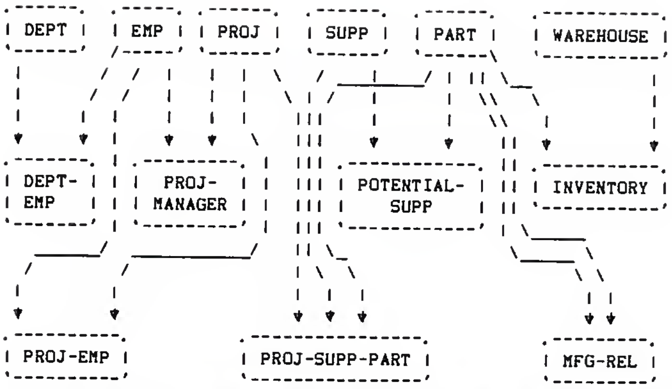
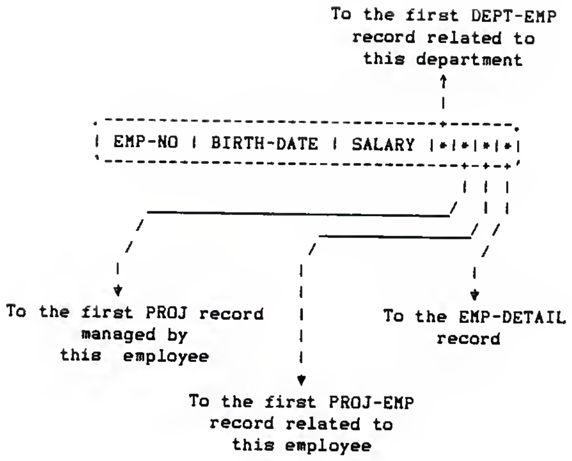


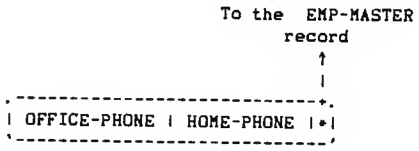
Figure 4.35 Another data-structure diagram derived from the E-R diagram in Figure 4.31a. (Adapted from [Chen 85])

mapping. This type of mapping change will convert a data-structure set into a record type or vice versa, based on the translation rules discussed in Section 4.2.1, but no change is necessary if the simplified rule discussed here is used.

A modification of the rules for translating E-R diagrams into data-structure diagrams will provide better database performance or better utilization of storage space. The EMP record shown in Figure 4.27, 4.29, and 4.35, can be split into two records. The EMP-MASTER record contains the fields EMP-NO, BIRTH-DATE, and SALARY (Figure 4.36a). The EMP-DETAIL record contains the fields STARTING-DATE-IN-DEPT, OFFICE-PHONE, and HOME-PHONE (Figure 4.36b). Pointers are added that connect the occurrence of the the EMP-MASTER and the EMP-DETAIL records. The data-structure diagrams in


To the first DEPT-EMP record related to this department
EMP-NO
BIRTH-DATE
SALARY
*
To the first PROJ record managed by this employee
To the EMP-DETAIL record
To the first PROJ-EMP record related to this employee

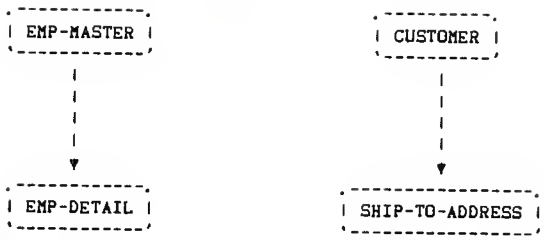
(a) EMP-MASTER record



(b) EMP-DETAIL record

Figure 4.36 (Adapted from [Chen 85])

Figure 4.27 and 4.35 would be modified by incorporating Figure 4.37. One of the reasons for splitting a record into two or more records would improve retrieval performance. It may be expected that the fields in the EMP-MASTER record will be used more often than those in the EMP-DETAIL record. Since retrieving unnecessary data is not a beneficial aspect of a DBMS the splitting



(a) Data-structure diagram for EMP-MASTER and EMP-DETAIL.

(b) A data-structure diagram for CUSTOMER and SHIP-TO-ADDRESS.

Figure 4.37 (Adapted from [Chen 85])

of a record may be justified. Another reason for splitting a record is a limitation of record size, due to hardware or software limitations. If the "conceptual" record is larger than the maximum length of a record, the "conceptual" record may have to be split into two or more records.

Another common practice for increasing performance is to factor out repeating groups. SHIP-TO-ADDRESSES for example, is a repeating group in a customer record (i.e., there are many data values for this attribute). This field can be moved out and be placed into a new record called SHIP-TO-ADDRESS (Figure 4.37b).

An E-R diagram may be translated into many different data-structure diagrams depending on different processing needs. Therefore the database designer should start with an E-R diagram and then translate it into a data-structure diagram suitable for the particular DBMS implementation.

4.5 HIERARCHICAL DATABASE DESIGN

Data in hierarchical database systems is organized into hierarchies of records which only allow 1:N mappings. Relationship types with M:N mappings must be translated into hierarchical structures. There are at least five possible logical data structures for the E-R diagram in Figure 4.38a, that require translation:

1. The PROJ record type in Figure 4.38a may be treated as a "child-record" for the EMP record type in Figure 4.38b. This logical data structure will be efficient for certain types of queries but not for others. If a search for the employees associated with a particular project was done there may have to be an exhaustive search of the entire database.

2. The EMP record type in Figure 4.38a may be treated as a "child-record" for the PROJ record type in Figure 4.38c. If a search for all the projects associated with a particular employee were needed, as before an exhaustive search of the entire database would be needed.

3. Since the logical data structures in Figure 4.38b or Figure 4.38c aren't efficient for all types of queries, two databases may have to be maintained as shown in Figure 4.39a. But this forces storage and maintenance of redundant data.

4. For a hierarchical database system, the logical data structure in Figure 4.39b may be chosen so that the EMP record type will be the "physical parent" of PROJ-EMP, and the PROJ record type will be the "logical parent."

5. An alternative, in a hierarchical database system, is to make the EMP record type the "logical parent" instead of the "physical parent" of PROJ-EMP record type (Figure 4.39c).

```
                   WORK-FOR
.-----. M                    N .------.
| EMP |-----------------------| PROJ |
'-----'                        '------'
```

(a) Many-to-many mapping

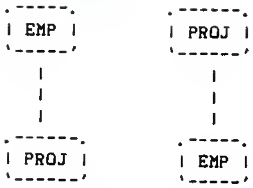
```
 .-----.
 | EMP |
 '-----'
    |
    |
    |
 .------.
 | PROJ |
 '------'
```

(b) PROJ as a child-record for EMP

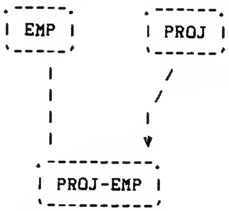
```
 .------.
 | PROJ |
 '------'
    |
    |
    |
 .-----.
 | EMP |
 '-----'
```

(c) EMP as a child-record for PROJ

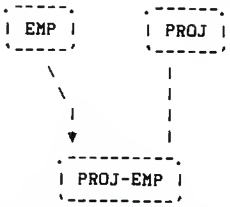
Figure 4.38 (Adapted from [Chen 85])



(a) Maintaining two databases.



(b) PROJ as the "logical parent" of PROJ-EMP



(c) EMP as the "logical parent" of PROJ-EMP

Figure 4.39 (Adapted from [Chen 85])

The Entity-Relationship Approach to logical database design has attracted considerable attention in industry and the research community. Many people have used this approach in the real-world environment and have found it easy to understand and to use. The E-R diagrammatic technique has also been found to be an effective tool between the end-users of the database and its designers for the specifications of user information requirements. The E-R Approach is a practical approach for logical database design and it is a valuable tool for the initial design where simplicity of technique is required.

Chapter 5

EMYCIN

The first step in defining an Expert Assistant that assists database design is to evaluste the reasons for choosing s psrticular Expert System design tool. Since most of the tools svsilable are not designed for any particular clsss of problems the selection process may become difficult. For every Expert System design tool there is a problem task suited to it [Waterman 86], the converse of this is not true. For any given problem task there may be seversl tools that could possibly be used. To simplify the decision process the sophisticstion, the support fscilities, the reliability, the msintenance, and the ussble features of the design tool must be examined.

The type of tool needed for assisting the Entity-Relationship approach to database design must have the following abilities:

Make decisions based on rules defined by [Chen 85],

Explain decisions msde, and

Store and access the experience of previous E-R users.

Chapter 3 explsined that the EMYCIN skeletal knowledge engineering system would provide the necessary sbilities to create the Expert Assistant for the E-R approach to database design. Following a description of EMYCIN, the Expert Assistant will be defined by formulsting the necesssry rules that will mske the Expert Assistant model viable, and a definition of how a Data-Dictionary could be automatically created.

EMYCIN is a programming system used to write knowledge-based consultation programs using production-rules to represent its knowledge. A domain independent version of the MYCIN Expert System, EMYCIN, developed at Stanford University as part of the Heuristic Programming Project [van Melle 79], uses a rule-based knowledge representation scheme with a rigid backward-chaining mechanism. EMYCIN has been used to build diagnosis-type Expert Systems in the areas of medicine, geology, engineering, agriculture, and other areas. Its facilities include an explanation program, a well-engineered environment for developing the knowledge base, and tracing and debugging programs. EMYCIN is best suited for deductive problems that are associated with large amounts of unreliable input data that has a specifiable solution space. This section presents the characteristics of EMYCIN regarding its knowledge representation, problem-solving knowledge, and other facilities.

5.1 KNOWLEDGE REPRESENTATION

EMYCIN's knowledge is represented using production rules written in LISP, and are comprised of a premise, which is formed by a conjunction of predicates over triples (attribute-object-value) in the knowledge base, and an action. If the premise is true the action or conclusion part of the rule is evaluated. If the premise is not known with enough certainty to be absolutely true, the strength of the conclusion is modified accordingly. Uncertainty in the data or competing hypothesis is represented by attaching a certainty factor to each triple. This certainty factor is usually a number between -1 (definitely false) to 1 (definitely true). The following is a typical rule from the domain of structural analysis [van Melle 79]:

```
If:   1) The material composing the sub-structure is one of
         the metals,
      2) The analysis error (in percent) that is  tolerable
         is less than 5,
      3) The non-dimensional stress of the sub-structure is
         greater than .5, and
      4) The  number of cycles the loading is to be applied
         is greater than 10000
Then:    It  is definite (1.0) that fatigue is one  of  the
         stress behavior phenomena in the sub-structure
```

Represented in LISP this rule appears as:

```
PREMISE: ($AND
              (SAME CNTXT COMPOSITION (LISTOF METALS))
              (LESSP* (VAL1 CNTXT ERROR) 5)
              (GREATERP* (VAL1 CNTXT ND-STRESS) .5)
              (GREATERP* (VAL1 CNTXT CYCLES) 10000))
ACTION:  (CONCLUDE CNTXT SS-STRESS FATIGUE TALLY 1.0)
```

Each rule is intended to provide a single piece of information, the knowledge base is therefore modular, in that it is relatively easy to update. Rules can be added deleted or modified without affecting the overall performance of the system. These rules are also useful for explanation purposes and since the system has the ability to read its own rules, the explanation program and other routines assisting the user are used extensively.

5.2 INFERENCE ENGINE

The control structure employed by EMYCIN is a goal-directed backward-chaining mechanism, the goal of which is to determine the action to take given the premise. At any time EMYCIN is attempting to work towards this goal by establishing the value of the action of some premise. To accomplish this, EMYCIN retrieves a list of rules whose conclusions are related to the goal, then systematically attempts to apply the rules. This application continues until the goal is satisfied with a given certainty, or the rule list has been exhausted. If other information is needed when the premises are evaluated, EMYCIN produces subgoals to find out the information, causing other rules to be used. If no value can be deduced, either because there were no related rules or evaluation of the rules was unsuccessful, the system queries the user for the missing values. When the user cannot supply the information, the data becomes unknown causing future rules that require it to fail.

5.3 FACILITIES

EMYCIN's explanation facility allows the user to understand the reasons why a particular conclusion was reached and to examine the system's knowledge base. Examining the knowledge base allows the user to discover information about inferences made in a particular case at hand and to examine the static knowledge base in general.

Responding to user commands, in this case WHY a question was asked by EMYCIN, or HOW EMYCIN reached a conclusion, EMYCIN can explain the current, past, and possible future lines of reasoning. The explanation program can also be useful for debugging the final developed system. This can be accomplished without manipulating the system at the LISP level, providing for examination of what inferences have been made, why others failed, and allowing for corrections of errors and omissions in the knowledge.

Knowledge acquisition constructs the rules in the knowledge base and the object-attribute structures upon which the rules operate. Rules can be entered into the system by using an Abbreviated Rule Language, a formal representation mechanism which is much more like English than LISP. Rules in the knowledge base are modified by a high-level editor which checks each rule for syntactic validity and insures that no contradictions exist. When a rule is created or updated, the date, time, and user responsible are recorded with the rule. Once properties have been given legal values they are used by the system to prompt for omitted values and to check for errors. Once a rule is entered into the knowledge base and checked for validity, data structures are updated such as the data structure responsible for telling the rule interpreter which rules conclude results about which premise.

EMYCIN has the capability of keeping and maintaining various problem scenarios in libraries that are used for testing a complete system or for debugging one being built. When a library routine is rerun, questions are answered by supplying a response that was given when the scenario was run initially. Many of these cases may be run in the background mode allowing the system to check current results with those already obtained. This type of processing allows new rules added to the knowledge base to be checked from previously proven results. These features greatly facilitate the development of a new system [Hayes-Roth 83].

Chapter 6

TAXONOMY OF THE EXPERT ASSISTANT

The Expert Assistant, designed using EMYCIN, will operate much the same as if an experienced user were peering over the shoulder of a database designer using the E-R approach. If a mistake is made the Expert Assistant will prompt the user accordingly. If decision assistance is needed, the user needs only to ask the Expert Assistant. Already knowing the history of the current design session, the Expert Assistant can give meaningful suggestions based on what current rules are active in the Inference Engine and the information found in the knowledge base.

EMYCIN's facilities allow the user to establish the validity of each explanation much like a human consultant does. Backward-chaining is best suited for applications when the solution to a problem begins with a small set of states to a larger set of states. This parallels the E-R database design process which has an initial state of designing the conceptual schema by identifying the entities and relationships which are of interest to the enterprise. Subsequent states become numerous and varied when this schema is translated into data structures for a particular database system. The knowledge acquisition facility of EMYCIN will allow the Expert Assistant designer to store the experience of previous users of the E-R approach to database design.

The Expert Assistant will perform the database design process in three modes. Modes one, DIAGRAM, and two, TRANSLATE, are based on the rules defined by [Chen 85]. The DIAGRAM mode will allow the user to define the entities and relationships that are deemed necessary to the enterprise. The TRANSLATE mode, based on the architecture of the enterprises' DBMS, translates the E-R diagrams into the data structures needed. After the TRANSLATE mode is completed, the final mode of operation creates a Data-Dictionary for future database users and will be fully automated based on the requirements found in [Cardenas 79]. The TRANSLATE mode will be executed by the Expert Assistant without intervention of the user. Since once the architecture is known, the translation is straight forward. The user may stop this automatic process in order to monitor the progress of the Expert Assistant and to request information as to why or how a particular result was obtained.

The Expert Assistant will incorporate all of the rules applicable to the Entity-Relationship Approach to database design as defined by [Chen 85]. Representation of the rules is a straight forward process since they are conditional with a left hand side versus a right hand side. For example, a free-form English format of a rule for creating a composite entity might be represented as:

```
IF:   A relation might have attributes > .5
THEN: It is definite (1.0) that the relation should be
      converted to a composite entity.
```

And represented as an actual EMYCIN rule as:

```
    PREMISE: (GREATERP (VAL1 ATTRIBUTE (LISTOF RELATIONS)) .5)
     ACTION: (CONCLUDE RELATION CONVERT COMPOSITE TALLY 1.0)
```

This thesis presents the rules of the Expert Assistant based on the "terse" rule format defined by [van Melle 79] (see Appendix 3). This format is a simplified language used to bridge the gap between rules in free-form English text format (see Appendix 2) and LISP input. In the "terse" rule format, the rule for diagramming a composite entity (shown above) might look like:

```
     If Relation = (LISTOF ATTRIBUTES)
Then Composite Entity = RELATION
```

Appendix 1 defines the objects that are used in the rules of the Expert Assistant. The following sections describe the objects and the rules which the Expert Assistant will operate on.

6.1 OBJECTS

The DATABASE-LIST will contain the name of the database that is or was designed by the E-R Approach. The ENTITY-LIST contains each ENTITY and its type. The RELATION-LIST has each RELATION identifier, its type, and any dependencies. This list will also contain the entities that are involved with each particular relation. The ENTITY, its properties i.e., ATTRIBUTE-VALUE pairs including the identifier, the value(s), and the type, are found in the ATTRIBUTE-LIST.

The translation rules require that there be a list of owner records, member records, data structures, and pointers. The OWNER-

1. Create a unique entity name and its' type.
2. Create the attribute and value information for the current entity.
3. Create a unique relation name, type and dependency.
4. Create the other entity involved in the current relation.

Figure 6.1 Design steps facilitating the use of the Expert Assistant.

RECORD-LIST will contain the SET-NAME and the component ENTITY found in a relation. The MEMBER-RECORD-LIST contains the other ENTITYs and the SET-NAME involved in a relation. A list of all the SET-NAMEs found in a database are stored in the DATA-STRUCTURE-LIST. And finally, each member and owner record with their respective pointers are located in the POINTER-LIST.

The DATABASE-RECORD-LIST, FIELD-ATTRIBUTE-LIST, and the RECORD-FIELD-LIST contain the necessary information for the data-dictionary, and will be stored in the DATA-DICTIONARY-LIST. The database names and all the records contained in each database are located in the DATABASE-RECORD-LIST. Each field name and its attribute(s) are contained in the FIELD-ATTRIBUTE-LIST, and each record and its corresponding fields are stored in the RECORD-FIELD-LIST.

6.2 DEFINITION OF RULES

Based on the defined rules, the Expert Assistant model requires that the designer create the E-R diagrams by following the steps

```
RULE 1:: Obtain MODE


RULE 2:: IF:    MODE is DIAGRAM
         THEN:  Determine DATABASE


RULE 3:: IF:    MODE is TRANSLATE
         THEN:  Determine Data-structures


RULE 4:: IF:    Determining DATABASE
         THEN:  1) Obtain DATABASE
                and
                2) Obtain DBMS ARCHITECTURE


RULE 5:: IF:    DATABASE is in DATABASE-LIST
         THEN:  Report Error and Stop Expert Assistant


RULE 6:: IF:    DATABASE is not in DATABASE-LIST
         THEN:  Determine ENTITY
```

Figure 6.2 Rules composing the front end to the Expert Assistant.

shown in Figure 6.1. RULE 1 through RULE 6 (see Figure 6.2) limit the type of operations allowed and start the Expert Assistant in its monitoring process. The RULEs are based on the premise/action protocol needed by EMYCIN. When MODE, in RULE 1, becomes true the premise of RULE 2 and RULE 3 become true, causing the actions (the THEN clauses) to be evaluated. The action part of RULE 2 causes the premise of RULE 4 to become true which obtains the DATABASE its ARCHITECTURE. The action of RULE 4 instantiates DATABASE, causing the premise of RULE 5 or RULE 6 to become true, etc.

```
RULE D1:     IF:   Determining ENTITY
             THEN: Obtain ENTITY
                   ; i.e. (WHILE being entered)


RULE D2::    IF:   The ENTITY is in ENTITY-LIST
             THEN: Report Duplicate ENTITY
                   Determine RELATION


RULE D3::    IF:   The ENTITY is not in ENTITY-LIST
             THEN: 1) Determine ENTITY type
                   and
                   2) Determine RELATION
```

Figure 6.3 Rules for incorporating an ENTITY into the database design.

6.3 DIAGRAMMING RULES

The Diagramming rules are based on the descriptions of E-R diagrams discussed in Section 4.1. As shown in Figure 6.3, RULE D1 through RULE D3 are used to monitor the creation of the ENTITYs of interest to the enterprise. RULE D1 monitors entities as they are created by the designer (see Figure 6.1 step 1). The THEN clause of this rule instantiates ENTITY, firing RULE D2 insuring that this entity does not already exist. If this entity does exist, after informing the user of this duplication, the Expert Assistant assumes that this entity will become the component of a new relation. The action part of RULE D1 also fires RULE D3 which establishes the ENTITY's type and the relation concerning it, as described in Section 4.1.2.

```
RULE D4::  IF:    Determining ENTITY Type
           THEN:  1) Obtain type
                     ; i.e. (Binary, Composite, or Elementary)
                  and
                  2) add ENTITY and type to ENTITY-LIST
                  and
                  3) Determine ATTRIBUTE-VALUE


RULE D5::  IF:    Determining RELATION
           THEN:  1) Obtain Unique RELATION identifier
                  and
                  2) Obtain RELATION type
                     ; i.e. (1:1, 1:N, or M:N)
                  and
                  3) Obtain Dependency
                     ; i.e. (None, Existent, or ID)
                  and
                  4) Obtain Entity(s)
                  and
                  5) Determine Duplications
                  and
                  6) Obtain Entity Type
                  and
                  7) Add involved ENTITY(s), RELATION
                     identifier, type, and dependency
                     to RELATION-LIST
```

Figure 6.4 Rules used to determine the ENTITY type and the RELATION of the current ENTITY.

RULE D3 fires RULE D4 and RULE D5 (Figure 6.4) establishing the necessary information about the current entity. Based on the description of entities in Section 4.1.1 and Section 4.1.3, The first action in RULE D4 is to retrieve the ENTITY type created by the user according to Step 1 in Figure 6.1. The second action adds the entity to the list of entities contained in the current DBMS design, and the third action fires RULE D6 (Figure 6.5) which will determine the properties of the current ENTITY. Action 1, 2, and 3 in RULE D5 retrieves a unique RELATION identifier, RELATION

```
RULE D6::  IF:    Determining ATTRIBUTE-VALUE
           THEN:  1) Obtain Unique ATTRIBUTE-VALUE identifier
                  and
                  2) Obtain ATTRIBUTE-VALUE type
                     ; i.e. (1:1 or 1:N)
                  and
                  3) Obtain Value(s)
                  and
                  4) Add identifier, ATTRIBUTE-VALUE, and type
                     to ATTRIBUTE-LIST
```

Figure 6.5 Rule to determine the properties of the current ENTITY.

type, and any dependencies (Section 4.1.2 and 4.1.6) deemed necessary by the designer according to Step 3 in Figure 6.1. Actions four through six, in RULE D5, retrieve the other entity(s) involved in the current relation following the same entity limitations as discussed before, and the seventh action adds this information to the list of relations contained in the current DBMS design.

RULE D6 (see Figure 6.5) fired by RULE D4 in Figure 6.4, insures that the necessary properties are recorded about the current ENTITY as described in Section 4.1.4. As with relations, the properties of an ENTITY (i.e., ATTRIBUTEs and VALUEs) must be given a unique identifier and a type and is accomplished by the first two actions in RULE D6. The third action retrieves the value or values created by the designer related to the current ENTITY. And the last action of RULE D6 adds this information to a list containing the same type of information about the other ATTRIBUTE-VALUEs in the current design.

```
RULE T1::  IF:   Mode is TRANSLATE
           THEN: 1) Parse RELATION-LIST
                 and
                 2) Implement Data-structure set
                 and
                 3) Add information to DATABASE-LIST
                 and
                 4) Process DATA-DICTIONARY
                 and
                 5) Add information to
                    DATA-DICTIONARY-LIST


RULE T2::  IF:   Parsing RELATION-LIST
                 ; while relations exist
           THEN: Obtain RELATION type
                 ; from RELATION-LIST
```

Figure 6.6 Rules comprising the front end to the translation mode.

If the current state of the Expert Assistant does not agree with what the database designer is attempting to accomplish, the system will prompt the designer. For example, if the system is expecting a relation to be entered for the current entity (RULE D5 in Figure 6.4) and this is not done, the system requests the necessary information to fulfill the rules. Likewise, if the designer does not know what to do next he may request assistance by asking HOW, and the system will reply by informing the user what is expected next. If the system initiates a prompt the user may ask for clarification by asking WHY, forcing the Expert Assistant to explain by chaining backward through the active rules.

```
RULE T3::  IF:   1) RELATION type is 1:1
                 or
                 2) RELATION type is 1:N
           THEN: 1) Create unique SET-NAME identifier
                 and
                 2) Add component ENTITY and SET-NAME
                    to OWNER-RECORD-LIST
                    ; from RELATION-LIST
                 and
                 3) Add second ENTITY and SET-NAME
                    to MEMBER-RECORD-LIST
                 and
                 4) Create RECORD(s)
                 and
                 5) Add ATTRIBUTE-VALUE(s) to RECORD
                 and
                 6) Add RECORD(s) to RECORD-LIST
```

Figure 6.7 Translation rule based for a 1:1 or 1:N relationship type.

6.4 TRANSLATION RULES

RULE T1 and RULE T2, in Figure 6.6, represent the front end of the Expert Assistants' TRANSLATE mode. RULE T1 starts creation of the data structures by parsing the ENTITY-LIST (created in RULE D4), and creates the DATA-DICTIONARY. Then RULE T1 implements the data structure sets and adds this information to the database. Finally the data-dictionary is created and this information is added to the DATA-DICTIONARY-LIST. Parsing in RULE T2, retrieves each piece of information in the RELATION-LIST and uses it to fire RULEs T3 through T5.

RULE T3 through RULE T6 are based on the translation RULEs for relationships defined on two entity types, three or more entity types, binary relationship types, and composite entity types

```
RULE T4::  IF:   1) RELATION type is M:N
                 and
                 2) ARCHITECTURE type is not Hierarchical
           THEN: 1) Create unique SET-NAME identifier
                 and
                 2) Add component ENTITIES and SET-NAME
                    to OWNER-RECORD-LIST
                 and
                 3) Translate relation to new member record
                 and
                 4) Add created record and SET-NAME
                    to MEMBER-RECORD-LIST
                 and
                 5) Create RECORD(s)
                 and
                 6) Add ATTRIBUTE-VALUE(s) to RECORD
                 and
                 7) Add RECORD(s) to RECORD-LIST


RULE T5::  IF:   1) RELATION type is M:N
                 and
                 2) ARCHITECTURE type is Hierarchical
           THEN: 1) Create first new 1:N OWNER-RECORD and
                    MEMBER-RECORD and types
                 and
                 2) Process 1:N RELATION
                 and
                 3) Create second new 1:N OWNER-RECORD and
                    MEMBER-RECORD and types
                 and
                 4) Process 1:N RELATION
```

Figure 6.8 Translation rules for a M:N relationship type.

presented in Section 4.2.1. RULE T3, in Figure 6.7, is used if the current relation type is a 1:1 or 1:N. The data set name in action one comprises the link between the owner record and the member record and must be unique. The component entity in any relation becomes the owner record of any new data structure, while the other relationally involved entity becomes the member record type. This method of defining owner and member records is used in all of the rules found in Section 4.2.1. The information concerning the owner and member record types are stored in lists (action two, three, and six), facilitating the creation of a data-dictionary. In action four, records are created from the information already obtained in this rule, and are given their fields (variables) by retrieving the pertinent information from the ATTRIBUTE-VALUE-LIST (RULE D6 in Figure 6.5).

RULE T4 and RULE T5 in Figure 6.8, are similar to RULE T3 except that the type of DBMS architecture requires different processing. As discussed in Section 4.5, hierarchical and CODASYL DBMSs do not allow for M:N relationships. RULE T4 is used if the architecture is not this type of architecture and RULE T5 is used if the architecture is. As discussed in rule 1b in Section 4.2.1, a M:N mapping type is converted by making a new entity from the relation name with two arrows pointing from the related entity record types. Action one in RULE T4 accomplishes this by creating unique set name identifiers, action two adds the component entities and the set names to the OWNER-RECORD-LIST. The newly created member record, action three, is added to the MEMBER-RECORD-LIST as in RULE T3.

Although not the most efficient method, for simplicity RULE T5 (in Figure 6.8) creates two separate 1:N relations to facilitate allowable data structure sets and records to be defined for a hierarchical DBMS architecture. Action one and two create the new 1:N OWNER-RECORD and MEMBER-RECORD, their respective types (from

```
RULE T6::  IF:    Implementing Data-structure set
                  ; i.e. (from RECORD-LIST)
           THEN: 1) Add SET-NAME to DATA-STRUCTURE-LIST
                  and
                  2) Parse OWNER-RECORD-LIST
                  and
                  3) Parse MEMBER-RECORD-LIST


RULE T7::  IF:    Parsing OWNER-RECORD-LIST
           THEN: Determine MEMBER-RECORD(s) pointer
                  ; i.e. using SET-NAME in OWNER-RECORD-LIST


RULE T8::  IF:    Parsing MEMBER-RECORD-LIST
           THEN: 1) Determine next MEMBER-RECORD if more
                     exist
                  ; i.e. from SET-NAME in MEMBER-RECORD-LIST
                  or
                  2) Determine OWNER-RECORD if no more exist
```

Figure 6.9 Rules used to begin creation of the pointers for a DBMS.

the original M:N relation), and are given a unique identifier. Action three adds the new relation and type to the RELATION-LIST, the new 1:N relations are added to the RELATION-LIST (action four and five) for later processing, and the old M:N relation is removed from the RELATION-LIST.

RULE T6 through T11 are used to generate the pointers needed to link the records in the DBMS. RULE T6 (in Figure 6.9), adds the current SET-NAME to the DATA-STRUCTURE-LIST in its first action. Action two and three invoke parsing of the owner and member record lists, RULE T7 and RULE T8, which is accomplished in much the same fashion as the ENTITY-LIST was parsed in RULE T2. Each RECORD is examined with these parses whether it is the first member record

```
RULE T9::  IF:    Determining MEMBER-RECORD(s) pointer
           THEN:  1) Obtain first MEMBER-RECORD
                  and
                  2) Create POINTER from OWNER-RECORD
                     to MEMBER-RECORD
                  and
                  2) Add OWNER-RECORD, POINTER, and
                     MEMBER-RECORD to POINTER-LIST


RULE T10:: IF:    Determining next MEMBER-RECORD if more
                  exist
           THEN:  1) Obtain next MEMBER-RECORD
                  and
                  2) Create POINTER from current MEMBER-RECORD
                     to the next MEMBER-RECORD
                  and
                  3) Add MEMBER-RECORD, POINTER, and
                     OWNER-RECORD to POINTER-LIST


RULE T11:: IF:    Determining next OWNER-RECORD if no more
                  MEMBER-RECORDs exist
           THEN:  1) Obtain OWNER-RECORD
                  and
                  2) Create POINTER from current MEMBER-RECORD
                     to the OWNER-RECORD
                  and
                  3) Add MEMBER-RECORD, POINTER, and
                     OWNER-RECORD to POINTER-LIST
```

Figure 6.10 Rules that create the pointers necessary to implement the previously created data structure sets.

for a owner record (RULE T8), or the next member record of the same owner record (RULE T9).

The rules in Figure 6.10 are used to establish the pointers necessary to implement the data structure sets created in the previous rules. RULE T9 establishes a pointer to the current owner records first member record found by using the SET-NAME in the

OWNER-RECORD-LIST. Action three then adds the OWNER-RECORD, the generated POINTER, and the MEMBER-RECORD to the POINTER-LIST. Member record pointers should indicate the next member record for the current owner record. However, the last member record should have a pointer directed to the owner record. RULE T10 obtains the next remaining MEMBER-RECORD for the current OWNER-RECORD and adds the MEMBER-RECORD, the created POINTER, and the OWNER-RECORD to the POINTER-LIST. If there does not exist more MEMBER-RECORDs in the MEMBER-RECORD-LIST, RULE T11 creates the pointer from the current MEMBER-RECORD to the OWNER-RECORD and stores the MEMBER-RECORD, its pointer, and the OWNER-RECORD in the POINTER-LIST.

6.5 DATA-DICTIONARY RULES

All the information recorded in the diagramming and translation mode is all that is necessary to create the Data-Dictionary. Knowing the DBMS architecture from the translation step, all that remains is to transform this information into the basic data structures that the Data-Dictionary requires. [Cardenas 79] listed some of the most common characteristics that a viable Data-Dictionary must have:

1) Lists of the database names and all the record names comprising each database.
2) Lists of the record names and all the data field names contained in each database.
3) Lists of field names and attributes of each field.

Other characteristics required for a viable Data-Dictionary, but can only be created once the system is being used are:

4) Lists of fields and the editing assigned to them.
5) Lists of record names and the password assigned to them.
6) Lists of field names and the password assigned to them.
7) Lists of field names and the names of all application programs which use each field.
8) Lists of system names and all application programs which comprise each system.
9) Lists of application program names and all field namea used in each program.
10) Lists of report names and all field names used in each report.
11) Lists of user names and all source document names controlled or received by each user.
12) Lists of user names and all report names controlled or received by each user.

RULE P1 through RULE P4 are the necessary rules to implement the first three of these requirements shown above. Information contained in the DATABASE-LIST, RECORD-LIST, and the ATTRIBUTE-LIST allow for the creation of the lists having the characteristics mentioned. The lists, and the information they contain, created by the data-dictionary rules are:

1) The DATABASE-RECORD-LIST will contain the names of each database linked to the records it contains,
2) The RECORD-FIELD-LIST will contain each record in a particular database tied to their attribute names,
3) The FIELD-ATTRIBUTE-LIST will contain each field name and the possible values for each field.

Action four of the then clause of RULE T1 in Figure 6.6, starts the creation of the DATA-DICTIONARY by executing RULE P1 in Figure 6.11. The action part of RULE P1 fires RULE P2, which parses the DATABASE-LIST. The first action of RULE P2 adds the current database name to the DATABASE-RECORD-LIST, and then the second action adds the records to the DATABASE-RECORD-LIST. Action

```
RULE P1::  IF:    Processing DATA-DICTIONARY
           THEN:  Parse DATABASE-LIST


RULE P2::  IF:    Parsing DATABASE-LIST
                  ; while databases exist
           THEN:  1) Add DATABASE to DATABASE-RECORD-LIST
                  and
                  2) Add RECORD to DATABASE-RECORD-LIST
                  and
                  3) Parse RECORD-LIST
                  and
                  4) Parse ATTRIBUTE-LIST
```

Figure 6.11 Rules that begin the creation of the Data-Dictionary.

three parses the RECORD-LIST, and action four parses the ATTRIBUTE-LIST both of which were created in the TRANSLATION mode.

RULE P3 and RULE P4 in Figure 6.12, create the remaining lists used by the Data-Dictionary. The first action adds the RECORD to the RECORD-FIELD-LIST followed by adding the field name to the RECORD-FIELD-LIST. Action one of RULE P4 adds each field name encountered in the ATTRIBUTE-LIST to the FIELD-ATTRIBUTE-LIST. Action two then adds the values to the FIELD-ATTRIBUTE-LIST.

This Expert Assistant model fully encompasses the E-R Approach to database design, from defining the E-R diagrams through translating the diagrams into data structures for a DBMS and finally creating a minimal Data-dictionary. By placing the Expert Assistant in a monitoring mode the designer has the freedom to use a familiar tool without having to learn a new system. The Data-Dictionary gives the E-R approach an additional advantage. When

```
RULE P3::  IF:    Parsing RECORD-LIST
                  ; while records exist
           THEN: 1) Add RECORD to RECORD-FIELD-LIST
                  and
                  2) Add Field name to RECORD-FIELD-LIST


RULE P4::  IF:    Parsing ATTRIBUTE-LIST
                  ; while records-exist
           THEN: 1) Add Field name to FIELD-ATTRIBUTE-LIST
                  and
                  2) Add value to FIELD-ATTRIBUTE-LIST
```

Figure 6.12 Parsing rules used to create the Data-Dictionary.

the initial database is constructed, the designer may use the data-dictionary reporting facility (available for a particular DBMS), to insure that what was designed was actually built.

Chapter 7

APPLICATION OF THE EXPERT ASSISTANT

In order to validate the theoretical rules presented in Chapter 6, a minimal implementation was built that represents all of the protocols and requirements needed by the Expert Assistant (see Appendix 4). A simple enterprise with three of the entities shown in Figure 4.10a, was used to test this implementation, an employee entity, a project entity, and a department entity. The relationships between these entities is represented in Figure 7.1 as an Entity-Relationship Diagram, and in Figure 7.2 as a data-structure set implemented as chains.

Appendix 5 lists the output from a session using this implementation of the Expert Assistant rules. As described in Section 4.3, the major steps involved in logical database design is began by drawing an initial E-R diagram. Figure 7.3a illustrates that this is accomplished by entering "diagram" when the program is requesting "ENTER-MODE> ". The name of the database and its architecture is entered next, followed by entering all of the pertinent entity, relationship, and attribute-value information.

The "emp" entity is entered first along with its type, i.e. "e" representing an elementary type, and "c" representing a composite entity type (Figure 7.3b). The attribute-value for the "emp" entity as shown in Figure 7.1, and is identified by "ssn", and has a "1:1" attribute type, and a value of "123456789". The first

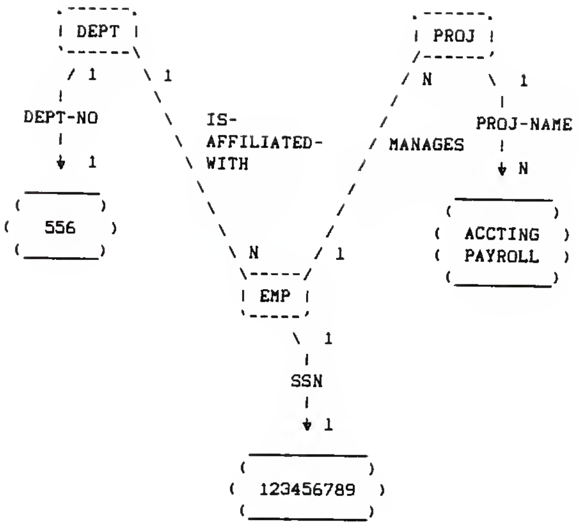


Figure 7.1 Simple E-R Diagram.

relation involved with the "emp" entity is "is-affiliated-with" and is a "1:N" relation with no dependency. Figure 7.3c illustrates how the entity involved with the "is-affiliated-with" relation and all of its information is entered into the system.

Figure 7.3d portrays how the second relation involved with the "emp" entity is entered. After all the necessary information is retrieved by the program for the first entities, the user is again prompted to enter an entity. If, at this time, the user enters an entity that already exists in the system, the program assumes that a second relation and entity is being defined for this entity and

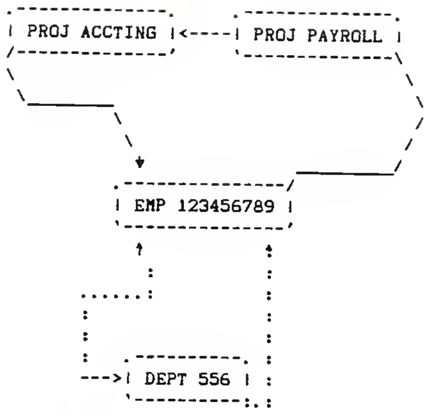


Figure 7.2 Implementation of the data-structure sets of E-R Diagram in Figure 7.1.

prompts the user with "(DUP-ENTITY ENTER NEW RELATION)". If this occurs the new relation "manages" for the "emp" entity is entered as was accomplished with the previously discussed "is-affiliated-with" relation. The necessary information for "proj" entity is entered followed by its attribute-value data. Since the attribute identifier "proj-name" has a "1:N" type, more than one value may be entered for this entity's attribute. After entering the "accting" and "payroll" attribute-values, "end" is entered to inform the program that there does not exist any more attribute-value pairs for this entity.

When the diagramming session is completed the user enters "end" when the "ENTER-ENTITY> " prompt is encountered. The system then

```
XLISP version 1.5, Copyright (c) 1985, by David Betz
; loading "ea.lsp"
ENTER-MODE> diagram
ENTER-DATABASE> db1
ENTER-DBMS-ARCHITECTURE> network
ENTER-ENTITY> emp
ENTER-ENTITY-TYPE> e
```

(a)

```
(-------- FOR THE EMP ENTITY ---------)
INPUT-ATTRIBUTE-VALUE-IDENTIFIER> ssn
INPUT-ATTRIBUTE-TYPE> 1:1
ENTER-ATTRIBUTE-VALUE> 123456789
ENTER-UNIQUE-RELATION-IDENTIFIER> is-affiliated-with
ENTER-RELATION-TYPE> 1:N
ENTER-RELATION-DEPENDENCY> n
```

(b)

```
ENTER-SECOND-ENTITY> dept
ENTER-ENTITY-TYPE> e
(-------- FOR THE DEPT ENTITY ---------)
INPUT-ATTRIBUTE-VALUE-IDENTIFIER> dept-no
INPUT-ATTRIBUTE-TYPE> 1:1
ENTER-ATTRIBUTE-VALUE> 556
```

(c)

```
ENTER-ENTITY> emp
(DUP-ENTITY ENTER NEW RELATION)
ENTER-UNIQUE-RELATION-IDENTIFIER> manages
ENTER-RELATION-TYPE> 1:1
ENTER-RELATION-DEPENDENCY> n
ENTER-SECOND-ENTITY> proj
ENTER-ENTITY-TYPE> e
(-------- FOR THE PROJ ENTITY ---------)
INPUT-ATTRIBUTE-VALUE-IDENTIFIER> proj-name
INPUT-ATTRIBUTE-TYPE> 1:N
ENTER-ATTRIBUTE-VALUE> accting
ENTER-ATTRIBUTE-VALUE> payroll
ENTER-ATTRIBUTE-VALUE> end
```

(d)

Figure 7.3 Example of entering the entities, relationships, and attributes for a minimal implementation.

translates the diagram into data-structure sets and creates a data-dictionary. The information for each design session is stored in various lists as described in Appendix 1, and can be seen in Appendix 5. For documentation purposes the information stored in these lists is printed when the user exits the program. The DATABASE-LIST contains all the information contained in the lists generated by the "TRANSLATE" process and each node in the list contains:

1. The database name,
2. The database architecture,
3. The OWNER-RECORD-LIST,
4. The MEMBER-RECORD-LIST,
5. The DATABASE-RECORD-LIST,
6. The ATTRIBUTE-LIST,
7. The DATA-STRUCTURE-LIST, and
8. The POINTER-LIST.

The "PROCESSING DATA-DICTIONARY" step creates the DATA-DICTIONARY-LIST which contains the necessary information defined in Section 6.5, and is made up by:

1. The DATABASE-RECORD-LIST,
2. The RECORD-FIELD-LIST, and
3. The FIELD-ATTRIBUTE-LIST.

Although this implementation does not incorporate all of the mechanisms that an Expert Assistant designed with EMYCIN would have, it does represent the basic control structure. The certainty factors were purposely omitted at this stage since the model would have them incorporated with it when it is created with EMYCIN.

Chapter 8

EVALUATION AND REMARKS

This thesis has described an Expert Assistant to database design based on the Entity-Relationship Approach. This model relieves much of the responsibility that is placed on a designer using this approach. The Expert Assistant would assist the designer by controlling design constructs insuring that the final product truly reflects the enterprise that the database represents. The translation of the entity-relationship diagrams into data structure diagrams and sets maximizes the probability that the database will be viable. The assistance realized by creating the data-dictionary becomes apparent when the information it contains is used by the designer.

The integrity of the rules that [Chen 85] defined, are maintained throughout the logical database design insuring that with each step of the design process the optimal decisions are made. Since the Expert Assistant is designed based on the constructs of the EMYCIN Expert System development tool, experienced suggestions are available to the designer as well as explanations that can remove uncertainty.

8.1 FUTURE ENDEAVORS

Further study in the diagramming stage of the Expert Assistant would give the system the capability of graphically representing the Entity-Relationship Diagrams as they are being defined. This would enhance the users capabilities as the process of designing a database is being carried out. The data-dictionary creation process could be made to create specific structures needed for any DBMS by incorporating this knowledge in the Knowledge Base of the Expert Assistant. This type of improvement could also be used to actually develop constructs which would allow the designer to create reports and programs that would be used in the final DBMS.

Constructs that would allow for the collection of the existing knowledge gained by other users of the Entity-Relationship approach would also facilitate this model. This would truly make the model a tool that could be widely used for various applications. Formulation of knowledge collection is paramount for any Expert System being developed since any Expert System does not truly wield power in its area of expertise based on its abilities as a problem solver but on the knowledge that it possesses.

Selected Bibliogrsphy

Albsno, A., V. DeAntonellis, and A. Di Levs, "Computer-Aided Dstsbase Design: The Datsid Approsch" in Computer-sided dstsbase design the DATAID project, A. Albano, V. DeAntonellis, snd A. Di Levs, eds., Science Publishing Compsny, INC., New York, 1985, pages 1-13.

Brady, L.I., "A Universsl Relation Assumption Bssed on Entities snd Relstionships", in Proceedings of the 4th International Conference on Entity-Relationship Approsch, IEEE Computer Society Press, Msryland, 1985, psges 208-215.

Bragger, R.P., A. Dudler, J. Rebssmen, and C.A. Zehnder, "GAMBIT, An Intersctive Databsse Design Tool for Dats Structures, Integrity Constraints snd Transsctions", in Proceedings of the 1984 Internstional Conference on Dsta Engineering, IEEE Computer Society Press, Cslifornis, 1984, psges 399-407.

Brisnd, H., H. Hsbriss, I.F. Hue, snd Y. Simon, "Expert System for Trsnslsting sn E-R Diagrsm into Dstsbsses", in the 4th Internstionsl Conference on Entity-Relstionship Approach, IEEE Computer Society Press, Marylsnd, 1985, pages 199-206.

Bubenko, J., and O. Kollhsmmer, "CADIS - Computer-Aided Design of Informstion Systems", in Computer-Aided Informstion Systems, Anslysis snd Design, Bubenko, Lsngefors, snd Solvberg, eds., Lund, Sweden, 1972, psges 119-140.

Cardenss, A.F., "Dsta Bsse Mansgement Systems", Allyn snd Bacon, Inc., Massschusetts, 1979.

Cszin, J., R. Jscquart, and P. Michel, "The F1 Formslism An Extension of the Entity-Relstionship Model Using First Order Logic", in the 4th International Conference on Entity-Relstionship Approsch, IEEE Computer Society Press, Marylsnd, 1985, pages 216-223.

Chen, P.P., "The Entity-Relationship Model - Toward a Unified View of Dats", in ACM Transactions on Datsbase Systems, V.1, No.1, Msrch, 1976, pages 9-36.

Chen, P.P., "The Entity-relstionship model - A bssis for the enterprise view of dsta", in Proceedings of the Nstursl Computer Conference, V.46, AFIPS Press, 1977, pages 77-84.

Chen, P.P., "Applications of the Entity - Relationship Model", in Proceedings of Database Design Techniques 1: Requirements and Logical Structures, New York, 1978, pages 87-113.

Chen, P.P., "A Preliminary Framework for Entity Relations Models", in Entity-Relationship Approach to Information Modeling and Analysis, P.P. Chen, ed., North-Holland, Amsterdam/New York, 1983, pages 19-28.

Chen, P.P., "An Algebra for a Directional Binary Entity-Relationship Model", in Proceedings of the 1984 International Conference on Data Engineering, IEEE Computer Society Press, California, 1984, pages 37-40.

Chen, P.P., "Database Design Based on Entity and Relationship", in Principles of Database Design Volume 1 Logical Organizations, S. Bing Yao, ed., Printice-Hall, New Jersey, 1985, pages 174-210.

Chung, I., F. Nakamura, and P.P. Chen, "A Decomposition of Relations using the Entity-Relationship Approach", in Entity-Relationship Approach to Information Modeling and Analysis, P.P. Chen, ed., North-Holland, Amsterdam/New York, 1983, pages 149-171.

DeMarco, T., and A. Soceneanta, "Data Flow Structures for System Specification and Implementation", in Proceedings of the 1984 International Conference on Data Engineering, IEEE Computer Society Press, California, 1984, pages 356-361.

Dumpala, S.R., and S.K. Arora, "Schema Translation Using the Entity-Relationship Approach", in Entity-Relationship Approach to Information Modeling and Analysis, P.P. Chen, ed., North-Holland, Amsterdam/New York, 1983, pages 337-356.

Elmasri, R.A., and J.A. Larson, "A Graphical Query Facility for ER Databases", in the 4th International Conference on Entity-Relationship Approach, IEEE Computer Society Press, Maryland, 1985, pages 236-245.

Elmasri, R.A., and G. Wiederhold, "<u>GORDAS</u>: A Formal High-Level Query Language for the Entity-Relationship Approach", in Entity-Relationship Approach to Information Modeling and Analysis, P.P. Chen, ed., North-Holland, Amsterdam/New York, 1983, pages 49-70.

Farmer, D., R. King, and D. Myers, "A Tool for the Implementation of Databases", in Proceedings of the 1984 International Conference on Data Engineering, IEEE Computer Society Press, California, 1984, pages 386-393.

Ferrera, F.M., "<u>EASY ER</u> An Integrated System for the Design and Documentation of Database Applications", in the 4th International Conference on Entity-Relationship Approach, IEEE Computer Society Press, Maryland, 1985, pages 104-113.

Fry, J.P., and T.J. Teorey, "Design & Performance Tools for Improving Database Usability & Responsiveness", in Databases: Improving Usability and Responsiveness, B. Shneiderman, ed., Academic Press, Inc., New York, 1978, pages 151-189.

Gane, C.P., and T. Sarson, "A Structural Systems Analysis: Tools and Techniques", Improved Systems Technologies, New York, 1977.

Han, S., and J.W. Cho, "KPSP: A Knowledge Programming System Based on Prolog", in the 4th International Conference on Entity-Relationship Approach, IEEE Computer Society Press, Maryland, 1985, pages 2-9.

Hartman, W., H. Matthes, and A. Proeme, "Management Information Systems Handbook - ARDI", McGraw-Hill Book Company, New York, 1968.

Hawryszkiewycz, I.T., "A Computer-Aid for E-R Modeling", in the 4th International Conference on Entity-Relationship Approach, IEEE Computer Society Press, Maryland, 1985, pages 64-69.

Hayes-Roth, F., and D.A. Waterman, and D. Lenat, eds., "Building Expert System", Addision-Wesley Publishing Company, Inc., Massachusetts, 1983.

Hwang, H.Y., and U. Dayal, "Using the Entity-Relationship Model for Implementing Multi-Model Database Systems", in Entity-Relationship Approach to Information Modeling and Analysis, P.P. Chen, ed., North-Holland, Amsterdam/New York, 1983, pages 235-256.

Jones, M.N., "HIPO for Developing Specifications", Datamation, March 1975, pages 112-125.

Kahn, B.K., "Requirement Specification Techniques", in Principles of Database Design Volume 1 Logical Organizations, S. Bing Yao, ed., Printice-Hall, New Jersey, 1985, pages 1-65.

Katzan, J., Jr., "System Design and Documentation: An Introduction to the HIPO Method", Van Nostrand Reinhold Co., New York, 1979.

Komorowski, H.J., "Rapid Software Development in a Database Framework - A Case Study", in Proceedings of the 1984 International Conference on Data Engineering, IEEE Computer Society Press, California, 1984, pages 394-398.

Konsynski, B.R., and Mannino, M., "Information Resource Specification and Design Language", in Proceedings of the International Conference on Entity-Relationship Approach to Systems Analysis and Design, P.P. Chen, ed., 1979, pages 346-358.

Hsu, C., "Structured Database System Analysis and Design Through Entity-Relationship Approach", in the 4th International Conference on Entity-Relationship Approach, IEEE Computer Society Press, Maryland, 1985, pages 56-63.

Lee, E.T., "Applications of Entity-Relationship Approach to Similarity-Driven Pictorial Database Design", in the 4th International Conference on Entity-Relationship Approach, IEEE Computer Society Press, Maryland, 1985, pages 18-21.

Ling, T.W., "A Normal Form For Entity-Relationship Diagrams", in the 4th International Conference on Entity-Relationship Approach, IEEE Computer Society Press, Maryland, 1985, pages 24-35.

Melkanoff, M.A., and C. Zaniolo, "Decomposition of Relations and Synthesis of Entity-Relationship Diagrams", in Proceedings of the International Conference on Entity-Relationship Approach to Systems Analysis and Design, P.P. Chen, ed., 1979, pages 285-302.

Merrett, T.H., "Relational Information Systems", Reston Publishing Co., Virginia, 1984.

Morgenstern, M., "A Unifying Approach for Conceptual Schema to Support Multiple Data Models", in Entity-Relationship Approach to Information Modeling and Analysis, P.P. Chen, ed., North-Holland, Amsterdam/New York, 1983, pages 279-297.

Orlando, S., P. Rullo, D. Sacco, and W. Staniszkis, "Integrated tools for Physical Database Design in CODASYL Environment", in Computer-aided database design the DATAID project, A. Albano, V. DeAntonellis, and A. Di Leva, eds., Science Publishing Company, INC., New York, 1985, pages 131-153.

Qian, X., and G. Wiederhold," Data Definition Facility of CRITIAS", in the 4th International Conference on Entity-Relationship Approach, IEEE Computer Society Press, Maryland, 1985, pages 46-55.

Ruoff, K.L., "Practical Application of IDEF1 as a Database Development Tool" in Proceedings of the 1984 International Conference on Data Engineering, IEEE Computer Society Press, California, 1984, pages 408-415.

Sernadas, A., and C. Sernadas, "The Use of E-R Abstraction for Knowledge Representation", in the 4th International Conference on Entity-Relationship Approach, IEEE Computer Society Press, Maryland, 1985, pages 224-231.

Staley, S.M., and Anderson, D.C., "Executable E-R Specifications for Database Schema Design", in the 4th International Conference on Entity-Relationship Approach, IEEE Computer Society Press, Maryland, 1985, pages 160-169.

Turk, T.A., "Planning and Designing the Data Base Environment", Van Nostrand Reinhold Company, Inc., New York, 1985.

van Melle, W., "A Domain-Independent Production-Rule System for Consultation Programs", in the Proc. of the Sixth International Joint Conference on Artificial Intelligence, pages 923-925.

Waterman, D.A., "A Guide to Expert Systems", Addision-Wesley Publishing Company, Inc., Massachusetts, 1986.

Wiederhold, G., "Database Design", Second Edition, McGraw-Hill Book Company, New York, 1983.

Yao, S.B., "Principles of Database Design Volume 1 Logical Organizations", S. Bing Yao, ed., Printice-Hall, New Jersey, 1985.

APPENDIX 1

OBJECTS

NAME	DEFINITION
ARCHITECTURE	Type of DBMS to be implemented e.g. network, hierarchical, or relational.
ATTRIBUTE-LIST	A list containing the unique name of the attribute, its' value(s) and the type of ATTRIBUTE-VALUE i.e. (1:1 or 1:N).
ATTRIBUTE-VALUE	The value(s) of the ATTRIBUTE-VALUE obtained from the user.
DATABASE	Name of the current database.
DATABASE-LIST	A list of the Databases used by the enterprise.
DATABASE-RECORD-LIST	A list of the Databases names and all of its records.
DATA-DICTIONARY-LIST	A list of all the information for the data dictionary.
DATA-STRUCTURE-LIST	A list of all the data structure sets (what owner record has what member records).
ENTITY	The current entity, whether it be elementary, binary, or composite.
ENTITY-LIST	A list of all the entities in the current design.
FIELD-ATTRIBUTE-LIST	A list of field names and the attributes of the fields for each record.
MEMBER-RECORD	A record stored in the MEMBER-RECORD-LIST that is a member in a particular Data-Structure set.

MEMBER-RECORD-LIST	A liat of sll the member records in the current design.
MODE	The current opersting mode of the model, i.e. (DIAGRAM, TRANSLATE)
OWNER-RECORD	A record stored in the OWNER-RECORD-LIST that ia sn owner in s particular dsta-structure aet.
OWNER-RECORD-LIST	A list of sll the owner records in the current design.
POINTER	Uaed to locate: The first member record for a owner record, the next member record of the current member record (if the exiats more member records in the aame data-atructure set), or the owner record of the current member record (if this is the last member record in this dats-atructure set).
POINTER-LIST	A list containing either sn owner record or a member record with its' respective pointer.
RECORD	A record crested for the finsl DBMS architecture.
RECORD-FIELD-LIST	A list of record names with esch of the correaponding field names.
RECORD-LIST	A list of all the records crested for the particular DBMS.
RELATION	A unique identifier that depicts the entities involved in s relation.
RELATION-LIST	A list containing the unique identifier for a relation, the type of relation, and the data dependency.
SET-NAME	A unique identifier thst depicta the owner snd member records involved in a dsta-atructure aet.

APPENDIX 2

FREE FORM ENGLISH TEXT FORMAT

```
RULE 1::    Obtain MODE


RULE 2::    IF:    MODE is DIAGRAM
            THEN:  Determine DATABASE


RULE 3::    IF:    MODE is TRANSLATE
            THEN:  Determine Data-structures


RULE 4::    IF:    Determining DATABASE
            THEN:  1) Obtain DATABASE
                   and
                   2) Obtain DBMS ARCHITECTURE


RULE 5::    IF:    DATABASE is in DATABASE-LIST
            THEN:  Report Error and Stop Expert Assistant


RULE 6::    IF:    DATABASE is not in DATABASE-LIST
            THEN:  Determine ENTITY
```

2.1 DIAGRAMMING RULES

```
RULE D1:    IF:    Determining ENTITY
            THEN:  1)Report Duplicate ENTITY
                   and
                   2) Obtain ENTITY
                   ; i.e. (WHILE being entered)


RULE D2::   IF:    The ENTITY is in ENTITY-LIST
            THEN:  Determine RELATION
```

```
RULE D3::  IF:   The ENTITY is not in ENTITY-LIST
           THEN: 1) Determine ENTITY type
                 and
                 2) Determine RELATION


RULE D4::  IF:   Determining ENTITY Type
           THEN: 1) Obtain type
                    ; i.e. (Binary, Composite, or Elementary)
                 and
                 2) add ENTITY and type to ENTITY-LIST
                 and
                 3) Determine ATTRIBUTE-VALUE


RULE D5::  IF:   Determining RELATION
           THEN: 1) Obtain Unique RELATION identifier
                 and
                 2) Obtain RELATION type
                    ; i.e. (1:1, 1:N, or M:N)
                 and
                 3) Obtain Dependency
                    ; i.e. (None, Existent, or ID)
                 and
                 4) Obtain Second Entity(s)
                 and
                 5) Determine Duplications
                 and
                 6) Obtain Entity Type
                 and
                 7) Add involved ENTITY(s), RELATION
                    identifier, type, and dependency
                    to RELATION-LIST


RULE D6::  IF:   Determining ATTRIBUTE-VALUE
           THEN: 1) Obtain Unique ATTRIBUTE-VALUE identifier
                 and
                 2) Obtain ATTRIBUTE-VALUE type
                    ; i.e. (1:1 or 1:N)
                 and
                 3) Obtain Value
                    ; Values if type is 1:N
                 and
                 4) Add identifier, ATTRIBUTE-VALUE, and type
                    to ATTRIBUTE-LIST
```

2.2 TRANSLATION RULES

```
RULE T1::  IF:    Mode is TRANSLATE
           THEN: 1) Parse RELATION-LIST
                 and
                 2) Implement Data-structure set
                 and
                 3) Add information to DATABASE-LIST
                 and
                 4) Process DATA-DICTIONARY
                 and
                 5) Add information to DATA-DICTIONARY-LIST


RULE T2::  IF:    Parsing RELATION-LIST
                 ; while relations exist
           THEN: Obtain RELATION type
                 ; from RELATION-LIST


RULE T3::  IF:   1) RELATION type is 1:1
                 or
                 2) RELATION type is 1:N
           THEN: 1) Create unique SET-NAME identifier
                 and
                 2) Add component ENTITY and SET-NAME
                    to OWNER-RECORD-LIST
                    ; from RELATION-LIST
                 and
                 3) Add second ENTITY and SET-NAME
                    to MEMBER-RECORD-LIST
                 and
                 4) Create RECORD(s)
                 and
                 5) Add ATTRIBUTE-VALUE(s) to RECORD
                 and
                 6) Add RECORD(s) to RECORD-LIST
```

```
RULE T4::  IF:    1) RELATION type is M:N
                  and
                  2) ARCHITECTURE type is not Hierarchical
           THEN:  1) Create unique SET-NAME identifier
                  and
                  2) Add component ENTITIES and SET-NAME
                     to OWNER-RECORD-LIST
                  and
                  3) Translate relation to new member record
                  and
                  4) Add created record and SET-NAME
                     to MEMBER-RECORD-LIST
                  and
                  5) Create RECORD(s)
                  and
                  6) Add ATTRIBUTE-VALUE(s) to RECORD
                  and
                  7) Add RECORD(s) to RECORD-LIST


RULE T5::  IF:    1) RELATION type is M:N
                  and
                  2) ARCHITECTURE type is Hierarchical
           THEN:  1) Create first new 1:N OWNER-RECORD and
                     MEMBER-RECORD and types
                  2) Process 1:N RELATION
                     and
                  3) Create second new 1:N OWNER-RECORD and
                     MEMBER-RECORD and types
                     and
                  4) Process 1:N RELATION


RULE T6::  IF:    Implementing Data-structure set
                  ; i.e. (from RECORD-LIST)
           THEN:  1) Add SET-NAME to DATA-STRUCTURE-LIST
                  and
                  2) Parse OWNER-RECORD-LIST
                  and
                  3) Parse MEMBER-RECORD-LIST


RULE T7::  IF:    Parsing OWNER-RECORD-LIST
           THEN:  Determine MEMBER-RECORD(s) pointer
                  ; i.e. using SET-NAME in OWNER-RECORD-LIST
```

```
RULE T8::  IF:    Parsing MEMBER-RECORD-LIST
           THEN: 1) Determine next MEMBER-RECORD if more exist
                 ; i.e. from SET-NAME in MEMBER-RECORD-LIST
                 or
                 2) Determine OWNER-RECORD if no more exist


RULE T9::  IF:    Determining MEMBER-RECORD(s) pointer
           THEN: 1) Obtain first MEMBER-RECORD
                 and
                 2) Create POINTER from OWNER-RECORD
                    to MEMBER-RECORD
                 and
                 2) Add OWNER-RECORD, POINTER, and
                    MEMBER-RECORD to POINTER-LIST


RULE T10::  IF:  Determining next MEMBER-RECORD if more
                 exist
           THEN: 1) Obtain next MEMBER-RECORD
                 and
                 2) Create POINTER from current MEMBER-RECORD
                    to the next MEMBER-RECORD
                 and
                 3) Add MEMBER-RECORD, POINTER, and
                    OWNER-RECORD to POINTER-LIST


RULE T11:: IF:   Determining OWNER-RECORD if no more
                 MEMBER-RECORDs exist
           THEN: 1) Obtain OWNER-RECORD
                 and
                 2) Create POINTER from current MEMBER-RECORD
                    to the OWNER-RECORD
                 and
                 3) Add MEMBER-RECORD, POINTER, and
                    OWNER-RECORD to POINTER-LIST
```

2.3 DATA-DICTIONARY RULES

```
RULE P1::  IF:   Processing DATA-DICTIONARY
           THEN: Parse DATABASE-LIST
```

```
RULE P2::  IF:    Parsing DATABASE-LIST
                  ; while databases exist
           THEN:  1) Add DATABASE to DATABASE-RECORD-LIST
                  and
                  2) Add RECORDs to DATABASE-RECORD-LIST
                  and
                  3) Parse RECORD-LIST
                  and
                  4) Parse ATTRIBUTE-LIST


RULE P3::  IF:    Parsing RECORD-LIST
                  ; while records exist
           THEN:  1) Add RECORD to RECORD-FIELD-LIST
                  and
                  2) Add Field name to RECORD-FIELD-LIST


RULE P4::  IF:    Parsing ATTRIBUTE-LIST
                  ; while records-exist
           THEN:  1) Add Field name to FIELD-ATTRIBUTE-LIST
                  and
                  2) Add value to FIELD-ATTRIBUTE-LIST
```

APPENDIX 3

TERSE RULE FORMAT

```
RULE 1::    Input MODE


RULE 2::    IF   MODE = 'DIAGRAM'
            THEN RULE 4
                      ; Determine DATABASE name


RULE 3::    IF   MODE = 'TRANSLATE'
            THEN RULE T1
                      ; TRANSLATE Diagrams


RULE 4::    INPUT DATABASE
            INPUT ARCHITECTURE
            RULE 5 or
            RULE 6


RULE 5::    IF   DATABASE = (LISTOF DATABASE-LIST)
            THEN Report Error and Stop Expert Assistant


RULE 6::    IF   DATABASE ^= (LISTOF DATABASE-LIST)
            THEN RULE D1
                      ; Input ENTITY(s)
```

3.1 DIAGRAMMING RULES

```
RULE D1:    INPUT ENTITY
            (WHILE being entered)
               RULE D2 or
               RULE D3
RULE D2::  IF    ENTITY = LISTOF(ENTITY-LIST)
           THEN  Report Duplicate ENTITY                          and
                 RULE D5
                    ; Determine RELATION


RULE D3::  IF    ENTITY ^= LISTOF(ENTITY-LIST)
           THEN  RULE D4                                          and
                    ; Input ENTITY type
                 RULE D5
                    ; Determine RELATION


RULE D4::        Input ENTITY type                                and
                    ; (Binary, Composite, or Elementary)
                 ENTITY-LIST = ENTITY-LIST + ENTITY               and
                 RULE D6
                    ; Determine ATTRIBUTE-VALUE


RULE D5::        Input unique identifier                          and
                    ; (for RELATION)
                 Input RELATION type                              and
                    ; (1:1, 1:N, or M:N)
                 Input Dependency                                 and
                    ; (None, Existent, or ID)
                 Input Second Entities                            and
                 Report Duplicate Entity                          and
                    ; (None, Existent, or ID)
                 RULE D4                                          and
                    ; (None, Existent, or ID)
                 RELATION-LIST = RELATION-LIST +
                    LISTOF(ENTITY(s), RELATION: identifier,
                           type, and dependency)


RULE D6::        Input unique identifier                          and
                    ; (ATTRIBUTE-VALUE)
                 Input ATTRIBUTE-VALUE type                       and
                    ; (1:1 or 1:N)
                 Input ATTRIBUTE-VALUE Value                      and
                    ; (1 or more- if 1:N)
                 ATTRIBUTE-LIST = ATTRIBUTE-LIST +
                    LISTOF(Identifier,
                           ATTRIBUTE-VALUE: type and value(s))
```

3.2 TRANSLATION RULES

```
RULE T1::          RULE T2                                          and
                      ; (Parse RELATION-LIST)
                   RULE T6                                          and
                      ; (Implement data-structure set)
                   DATABASE-LIST = DATABASE-LIST +
                      LISTOF(DATABASE, ARCHITECTURE,
                             OWNER-RECORD-LIST,
                             MEMBER-RECORD-LIST,
                             RECORD-LIST, ATTRIBUTE-LIST,
                             DATA-STRUCTURE-LIST,
                             POINTER-LIST)                          and
                   RULE P1                                          and
                      ; (Process DATA-DICTIONARY)
                   DATA-DICTIONARY-LIST =
                      DATA-DICTIONARY-LIST + LISTOF(
                      DATABASE-RECORD-LIST,
                      RECORD-FIELD-LIST +
                      FIELD-ATTRIBUTE-LIST)


RULE T2::          For each RELATION type in RELATION-LIST
                      RULE T3                                        or
                         ; (1:1 or 1:N RELATION)
                      RULE T4                                        or
                         ; (M:N RELATION and Hierarchical test)
                      RULE T5                                        or
                         ; (M:N RELATION)


RULE T3::     IF   RELATION type = 1:1 or
                   RELATION type = 1:N
            THEN   Input unique SET-NAME identifier                 and
                   OWNER-RECORD-LIST = OWNER-RECORD-LIST +
                      LISTOF(component ENTITY, SET-NAME)            and
                      ; from RELATION-LIST
                   MEMBER-RECORD-LIST = MEMBER-RECORD-LIST +
                      LISTOF(ENTITY(s), SET-NAME(s))                and
                   Create RECORD(s)                                 and
                   Add ATTRIBUTE-VALUE(s) to RECORD                 and
                   RECORD-LIST = RECORD-LIST + LISTOF(RECORD)       and
```

```
RULE T4::  IF   RELATION type = M:N and
                ARCHITECTURE type ^= 'Hierarchical'
           THEN Create unique SET-NAME identifier             and
                OWNER-RECORD-LIST = OWNER-RECORD-LIST +
                   LISTOF(component ENTITY, SET-NAME)         and
                Translate relation to new member record      and
                MEMBER-RECORD-LIST = MEMBER-RECORD-LIST +
                   LISTOF(created RECORD, SET-NAME)           and
                Create RECORD(s)                              and
                Add ATTRIBUTE-VALUE(s) to RECORD              and
                RECORD-LIST = RECORD-LIST + LISTOF(RECORD)    and


RULE T5::  IF   RELATION type = M:N and
                ARCHITECTURE type = 'Hierarchical'
           THEN Create first new 1:N OWNER-RECORD with
                   MEMBER-RECORD with types                   and
                RULE T3                                       and
                   ; (1:1 or 1:N RELATION)
                Create second new 1:N OWNER-RECORD with
                   MEMBER-RECORD with types                   and
                RULE T3                                       and
                   ; (1:1 or 1:N RELATION)


RULE T6::       DATA-STRUCTURE-LIST = DATA-STRUCTURE-LIST +
                   SET-NAME                                   and
                RULE T7                                       and
                   ; Parse OWNER-RECORD-LIST
                RULE T8                                       and
                   ; Parse MEMBER-RECORD-LIST


RULE T7::       For each member in OWNER-RECORD-LIST
                   RULE T9
                      ; Determine MEMBER-RECORD(s) pointer


RULE T8::       For each member in MEMBER-RECORD-LIST
                RULE T10                                       or
                   ; Determine next MEMBER-RECORD
                   ; if not the last member
                RULE T11                                       or
                   ; Determine next MEMBER-RECORD
                   ; if the last member
```

```
RULE T9::          Retrieve first MEMBER-RECORD for current
                      OWNER-RECORD                                   and
                         ; using SET-NAME in OWNER-RECORD-LIST
                   Create POINTER from OWNER-RECORD to
                      MEMBER-RECORD                                  and
                   POINTER-LIST = POINTER-LIST +
                      LISTOF(OWNER-RECORD, POINTER,
                             MEMBER-RECORD)


RULE T10:: IF      Current MEMBER-RECORD ^=
                      LAST-MEMBER(MEMBER-RECORD-LIST)
           THEN    Retrieve next MEMBER-RECORD                       and
                      ; from SET-NAME in MEMBER-RECORD-LIST
                   Create POINTER from current MEMBER-RECORD
                      to the next MEMBER-RECORD                      and
                   POINTER-LIST = POINTER-LIST +
                      LISTOF(MEMBER-RECORD, POINTER
                             OWNER-RECORD)


RULE T11:: IF      Current MEMBER-RECORD =
                      LAST-MEMBER(MEMBER-RECORD-LIST)
           THEN    Create POINTER from current MEMBER-RECORD
                      to the current OWNER-RECORD                    and
                   POINTER-LIST = POINTER-LIST +
                      LISTOF(MEMBER-RECORD, POINTER
                             OWNER-RECORD)
```

3.3 DATA-DICTIONARY RULES

```
RULE P1::  RULE P2


RULE P2::  For each DATABASE in DATABASE-LIST
              DATABASE-RECORD-LIST = DATABASE-RECORD-LIST +
                 DATABASE                                     and
              DATABASE-RECORD-LIST = DATABASE-RECORD-LIST +
                 RECORDs                                      and
              RULE P3                                         and
                 ; Parse RECORD-LIST
              RULE P4
                 ; Parse ATTRIBUTE-LIST


RULE P3::  For each RECORD in RECORD-LIST
              RECORD-FIELD-LIST = RECORD-FIELD-LIST +
                 RECORD                                       and
              RECORD-FIELD-LIST = RECORD-FIELD-LIST +
                 FIELD


RULE P4::  For each member in ATTRIBUTE-LIST
              FIELD-ATTRIBUTE-LIST = FIELD-ATTRIBUTE-LIST +
                 Field name                                   and
              FIELD-ATTRIBUTE-LIST = FIELD-ATTRIBUTE-LIST +
                 value                                        and
```

APPENDIX 4

IMPLEMENTATION OF RULES IN LISP

```
;
;:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
;:                                                        ::
;:  Expert Assistant Initialization                       ::
;:                                                        ::
;:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
;

(defun initea ()
    (setq database-record-list nil)
    (setq data-dictionary-list nil)
    (setq field-attribute-list nil)
    (setq data-structure-list nil)
    (setq member-record-list nil)
    (setq owner-record-list nil)
    (setq record-field-list nil)
    (setq attribute-list nil)
    (setq relation-list nil)
    (setq database-list nil)
    (setq pointer-list nil)
    (setq entity-list nil)
    (setq record-list nil)
 )


;
;:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
;:                                                        ::
;:  Exiting Functions                                     ::
;:                                                        ::
;:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
;

; Print lists

(defun print-lists (db-list)
   (cond ((eq db-list nil)
          (terpri)
          (terpri))
         (t(princ '---> )
           (print (car db-list))
           (print-lists (cdr db-list)))
    )
 )
```

```
; Output pertinent lists

(defun done ()

   (print 'database-record-list)
   (print-lists database-record-list)

   (print 'data-dictionary-list)
   (print-lists data-dictionary-list)

   (print 'field-attribute-list)
   (print-lists field-attribute-list)

   (print 'data-structure-list)
   (print-lists data-structure-list)

   (print 'member-record-list)
   (print-lists member-record-list)

   (print 'owner-record-list)
   (print-lists owner-record-list)

   (print 'record-field-list)
   (print-lists record-field-list)

   (print 'attribute-list)
   (print-lists attribute-list)

   (print 'relation-list)
   (print-lists relation-list)

   (print 'database-list)
   (print-lists database-list)

   (print 'pointer-list)
   (print-lists pointer-list)

   (print 'entity-list)
   (print-lists entity-list)

   (print 'record-list)
   (print-lists record-list)

 )
```

```
;
;::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
;:                                                            ::
;:  Necesssry Functions                                       ::
;:                                                            ::
;::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
;

; Atom Member Predicste

(defun memberp (to-find sesrch-list)
   (cond ((eq search-list nil) nil)
         ((member to-find (car sesrch-list)) t)
         (t (memberp to-find (cdr sesrch-list)))
    )
 )


; Retrieve Attributes for sn Entity

(defun get-sttribute (entity-val st-list)
   (cond ((eq entity-val (casr at-list))
          (list (csdr (csr st-list))
                (car (last (csr st-list)))))
         (t(get-attribute entity-val (cdr at-list)))
   )
 )


; Determine Whst an Entity Points to

(defun points-to (srec slist)
   (cond ((eq (cadr (car slist)) srec) (caar slist))
         (t(points-to srec (cdr slist)))
    )
 )


; Does S-set exist in the M-list

(defun more-exist (s-set m-list)
   (cond ((eq m-list nil) nil)
         ((eq (csr (cdar m-list)) s-set) t)
         (t(more-exist s-set (cdr m-list)))
    )
 )
```

```
;
;::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
;:                                                          ::
;:  Front End to Expert Assistant                           ::
;:                                                          ::
;::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
;

; RULE 2:: IF Mode is DIAGRAM

(defun rule2 (mode)
   (if (eq mode 'DIAGRAM)
       (rule4)
    )
 )

; RULE 3:: IF Mode is TRANSLATE

(defun rule3 (mode)
   (if (eq mode 'TRANSLATE)
       (rulet1)
    )
 )

; RULE 4:: Input DATABASE name and architecture

(defun rule4 ()
   (princ 'enter-database)
   (setq database (read))
   (princ 'enter-dbms-architecture)
   (setq architecture (read))
   (rule5)
   (rule6)
 )

; RULE 5:: DATABASE is in DATABASE-LIST

(defun rule5 ()
   (cond ((memberp database database-list)
          (print 'Existent-Database)
          (exit))
         (t)
    )
 )

; RULE 6:: DATABASE is not in DATABASE-LIST

(defun rule6 ()
   (ruled1)
 )
```

```
;
;::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
;:                                                                ::
;:  Diagramming Rules                                             ::
;:                                                                ::
;::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
;

; RULE D1:: Input ENTITYs

(defun ruled1 ()
   (prog ()
         loopd1
         (princ 'enter-entity)
         (setq entity (read))
         (cond ((not (eq entity 'END))
                (or (ruled2)
                    (ruled3)
                 )
                (go loopd1))
               (t)
          )
    )
 )


; RULE D2:: ENTITY is in ENTITY-LIST

(defun ruled2 ()
   (cond ((memberp entity entity-list)

          ; Report Duplicate entity
          (print (list 'dup-entity 'enter 'new 'relation))

          ; Determine Relation
          (ruled5))

         (t nil)
    )
 )
```

```
; RULE D3:: ENTITY is not in ENTITY-LIST

(defun ruled3 ()
   (cond ((not (memberp entity entity-list))

          ; Input Entity type
          (ruled4 entity)

          ; Determine Relation
          (ruled5))

         (t nil)
     )
 )


;   RULE D4:: Determining ENTITY type

(defun ruled4 (entity)

   (princ 'enter-entity-type)
   (setq entity-type (read))

   ; Add entity to entity-list
   (setq entity-list (cons (list entity entity-type)
                                 entity-list)
    )

   ; Determine Attribute-value
   (ruled6 entity)

 )


;   RULE D5:: Determining RELATION

(defun ruled5 ()

   ; Enter Relation name
   (princ 'enter-unique-relation-identifier)
      (setq relation (read))

   ; Enter relation type - 1:1, 1:N, or M:N
   (princ 'enter-relation-type)
      (setq relation-type (read))

   ; Enter dependency - None, Existent, or ID
   (princ 'enter-relation-dependency)
      (setq relation-dependency (read))
   ; Enter the Second entity for the current relation
   (prog ()
         loopd5
         (princ 'enter-second-entity)
         (setq second-entity (read))
         (cond ((memberp aecond-entity entity-list)
                (print (list 'no-duplicates-allowed))
                (go loopd5))
          )
    )

   ; Input Entity type
   (ruled4 aecond-entity)

   ; Add entitiea relation, relation-type, and
   ; relation-dependency to the relation-liat
   (aetq relation-list
      (cona (liat entity
                  aecond-entity
                  relation
                  relation-type
                  relation-dependency)
            relation-liat)
    )
 )


;   RULE D6:: Determining ATTRIBUTE-VALUE

(defun ruled6 (entity)

   (print (list '--------
                'For
                'the
                entity
                'entity
                '---------))

   ; Enter attribute-value name
   (princ 'Input-attribute-value-identifier)
      (aetq attribute-name (read))

   ; Enter attribute-value type - 1:1 or 1:N
   (princ 'Input-attribute-type)
       (aetq attribute-type (read))
   ; If the type of attribute is 1:N enter all values
   (cond ((eq attribute-type '1:N)
          (prog ()
                loopd6
                (princ 'enter-attribute-value)
                (setq attribute-value (read))
                (cond ((not (eq attribute-value 'END))
                       (setq attribute-list
                          (cons (list entity
                                      attribute-name
                                      attribute-type
                                      attribute-value)
                                 attribute-list)
                        )
                       (go loopd6))
                      (t)
                )
          ))

         ; Otherwise enter the only value
         (t(princ 'enter-attribute-value)
           (setq attribute-value (read))
           (setq attribute-list
              (cons (list entity
                          attribute-name
                          attribute-type
                          attribute-value)
                    attribute-list)
           ))
   )
)
```

```
;
;::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
;:                                                           ::
;:  Translation Rules                                        ::
;:                                                           ::
;::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
;


;   RULE T1:: If Mode is TRANSLATE assuming pointers

(defun rulet1 ()

   ; Parse relation list
   (rulet2 relation-list)

   ; Implement data-structure set from record list
   (rulet6)

   ; Add the database, architecture, owner record sets,
   ; member record sets, records, attributes,
   ; data structures, and pointers to the database list
   (setq database-list (cons (list database
                                   architecture
                                   owner-record-list
                                   member-record-list
                                   record-list
                                   attribute-list
                                   data-structure-list
                                   pointer-list)
                             database-list)
    )

   ; Process data dictionary
   (rulep1)

   ; Add the database, database record list, record field
   ; list, and the field attribute list to the
   ; data-dictionary list
   (setq data-dictionary-list (cons (list database
                                          database-record-list
                                          record-field-list
                                          field-attribute-list)
                                    data-dictionary-list)
    )

)
```

```
;   RULE T2:: For each RELATION type in RELATION-LIST

(defun rulet2 (r-list)

   (cond ((eq r-list nil) nil)

            ; Translate for 1:1 or 1:N relations
         (t(rulet3 (car r-list))

            ; Translate if the relation is M:N and the
            ; architecture is not hierarchical
            (rulet4 (car r-list))

            ; Translate if the relation is M:N and the
            ; architecture is hierarchical
            (rulet5 (car r-list))

            ; Translate the remaining relations
            (rulet2 (cdr r-list)))
    )
 )


;   RULE T3::  IF RELATION type is 1:1 or 1:N

(defun rulet3 (current-relation)

   (cond ((or (eq (cadr (cddr current-relation)) '1:1)
              (eq (cadr (cddr current-relation)) '1:N))

          ; Add the relation name and the data set
          ; to the list of owner records
          (setq set-name (gensym "SET"))
          (setq owner-record-list
              (cons (list (car current-relation) set-name)
                     owner-record-list)
           )
```

```
; Add the relation name and the data set
; to the list of member records
(prog (at-list)
      (setq at-list attribute-list)
      loopt4a
      (cond ((eq at-list nil))
            ((eq (cadr current-relation)
                 (caar at-list))
             (setq member-record-list
                (cons (list (cadr current-relation)
                            set-name)
                      member-record-list)
             )
             (setq at-list (cdr at-list))
             (go loopt4a))
            (t(setq at-list (cdr at-list))
              (go loopt4a))
       )
 )
```

```
         ; Add the owner record name and its attributes
         ; to the record list
         (setq record (gensym "OWNER-RECORD"))
         (setq record
             (cons record
                   (get-attribute (car current-relation)
                                   attribute-list))
          )
         (setq record-list (cons record record-list))

         ; Add the member record name and its attributes
         ; to the record list
         (prog (at-list)
               (setq at-list attribute-list)
               loopt4a
               (cond ((eq at-list nil))
                     ((eq (cadr current-relation)
                          (caar at-list))
                      (setq record-list
                        (cons (list (gensym "MEMBER-RECORD")
                                    (car
                                      (get-attribute
                                         (cadr
                                            current-relation)
                                          attribute-list))
                                    (car
                                       (last (car at-list))))
                              record-list)
                       )
                      (setq at-list (cdr at-list))
                      (go loopt4a))
                     (t(setq at-list (cdr at-list))
                       (go loopt4a))
                )
          ))
        (t)
   )
)
```

```
;   RULE T4::  IF RELATION type is M:N and not Hierarchical

(defun rulet4 (current-relation)

   (cond ((and (eq (cadr (cddr current-relation)) 'M:N)
               (not (eq architecture 'hierarchical)))

          ; Add the relation name and the data set
          ; to the list of owner records
          (setq set-name (gensym "SET"))
          (setq owner-record-list
              (cons (list (car current-relation) set-name)
                    owner-record-list)
           )

          ; Translate the relation to a new member record
          ; and add the relation name and the data set
          ; to the list of member records
          (setq member-record-list
              (cons (list (car (cddr current-relation))
                           set-name)
                    member-record-list)
           )

          ; Add the previous member record and the data set
          ; to the list of owner records
          (setq set-name (gensym "SET"))
          (setq owner-record-list
              (cons (list (cadr current-relation) set-name)
                    owner-record-list)
           )

          ; Add the relation name and the data set
          ; to the list of member records
          (setq member-record-list
              (cons (list (car (cddr current-relation))
                           set-name)
                    member-record-list)
           )

          ; Add the first owner record name and its
          ; attributes to the record list
          (setq record (gensym "OWNER-RECORD"))
          (setq record
              (cons record
                    (get-attribute (car current-relation)
                                   attribute-list))
           )
          (setq record-list (cons record record-list))
          ; Add the second owner record name and its
          ; attributes to the record list
          (setq record (gensym "OWNER-RECORD"))
          (setq record
              (cons record
                    (get-attribute (cadr current-relation)
                                   attribute-list))
           )
          (setq record-list (cons record record-list))

          ; Add the generated member record with no
          ; attributes to the record list
          (setq record (cons (car (cddr current-relation))
                             '(none))
           )
          (setq record (cons (gensym "MEMBER-RECORD")
                              record)
           )
          (setq record-list (cons record record-list)))
         (t)
    )
 )


;   RULE T5::  IF RELATION type is M:N and Hierarchical

(defun rulet5 (current-relation)

   (cond ((and (eq (cadr (cddr current-relation)) 'M:N)
               (eq architecture 'hierarchical))

          ; Generate a new 1:N relation and translate it
          (rulet3 (list (car current-relation)
                        (cadr current-relation)
                        (gensym "NEW-RELATION")
                         '1:N 'N)
           )

          ; Generate a second 1:N relation and translate it
          (rulet3 (list (cadr current-relation)
                        (car current-relation)
                        (gensym "NEW-RELATION")
                        '1:N 'N)
           )
          )
         (t)
    )
 )
```

```
;   RULE T6::  Implementing Data-structure set

(defun rulet6 ()

   (prog (o-list)
         (setq o-list owner-record-list)
         loopt6a
         (setq data-structure-list
               (cons (cdar o-list) data-structure-list)
          )
         (setq o-list (cdr o-list))
         (if (not (eq o-list nil)) (go loopt6a))
    )

   ; Parse the owner record list
   (rulet7 owner-record-list)

   ; Parse the member record list
   (rulet8 member-record-list)
 )


;   RULE T7:: Parsing OWNER-RECORD-LIST

(defun rulet7 (o-list)
   (cond ((eq o-list nil))

           ; Determine pointer to first member record
         (t(rulet9 (car o-list))

           ; Parse remainder of the owner records
           (rulet7 (cdr o-list)))
    )
 )


;   RULE T8:: Parsing MEMBER-RECORD-LIST

(defun rulet8 (m-list)
   (cond ((eq (cdr m-list) nil)(rulet11 (car m-list)))
         ((eq (more-exist (car (cdar m-list)) (cdr m-list)) t)

          ; Create pointers if more members exist create
          (rulet10 (car m-list))

          ; Parse remainder of the member records
          (rulet8 (cdr m-list)))

           ; Create pointer to the owner record if the
           ; last member record
         (t(rulet11 (car m-list))
          ; Parse remainder of the member records
          (rulet8 (cdr m-list)))
   )
 )


;   RULE T9:: Create owner record pointer to member record

(defun rulet9 (o-record)
   (setq pointer-list
      (cons (list (car o-record)
                  (gensym "POINTER")
                  (points-to (cadr o-record)
                      member-record-list))
             pointer-list)
   )
 )


;   RULE T10:: Create member pointer to the next member record

(defun rulet10 (m-record)
   (setq pointer-list
      (cons (list (car m-record)
                  (gensym "POINTER")
                  (points-to (cadr m-record)
                             member-record-list))
             pointer-list)
   )
 )


;   RULE T11:: Create member pointer to owner record

(defun rulet11 (m-record)
   (setq pointer-list
      (cons (list (car m-record)
                  (gensym "POINTER")
                  (points-to (cadr m-record)
                            owner-record-list))
              pointer-list)
   )
 )
```

```
;
;::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
;:                                                      ::
;:   Produce Data-Dictionary                            ::
;:                                                      ::
;::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
;

;   RULE P1:: Create Data-Dictionary

(defun rulep1 ()
   (terpri)
   (print (list '----------PROCESSING
                'DATA-DICTIONARY----------))
   (terpri)

   ; Add information to database record list for each
   ; database
   (rulep2 database-list)
 )


;   RULE P2:: Create Data-Dictionary

(defun rulep2 (d-list)
   (cond ((not (eq d-list nil))
          (setq database-record-list
             (cons record-list database-record-list)
           )
          (setq database-record-list
             (cons (caar d-list) database-record-list)
           )

          ; Parse record list
          (rulep3 record-list)

          ; Parse attribute list
          (rulep4 attribute-list)

          ; Process next database
          (rulep2 (cdr d-list)))
         (t)
    )
 )
```

```
;   RULE P3:: Parsing DATABASE-LIST

(defun rulep3 (r-list)
   (cond ((not (eq r-list nil))

          ; Add a record to the record field list
          (setq record-field-list
              (cons (car (cdar r-list)) record-field-list)
           )

          ; Add a field name to the record field list
          (setq record-field-list
             (cons (caar r-list) record-field-list)
           )

          ; Parse the remainder of the record list
          (rulep3 (cdr r-list)))
         (t)
    )
 )


;   RULE P4:: Parsing ATTRIBUTE-LIST

(defun rulep4 (a-list)
   (cond ((eq a-list nil))

            ; Add a field name to the field attribute list
         (t(setq field-attribute-list
              (cons (car (cddr (cdar a-list)))
                    field-attribute-list)
            )

          ; Add a value to the field attribute list
           (setq field-attribute-list
              (cons (car (cdar a-list)) field-attribute-list)
            )

           ; Parse the remainder of the attribute list
           (rulep4 (cdr a-list)))
    )
 )
```

```
;
;::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
;:                                                            ::
;:  Starting the Expert Assistant                             ::
;:                                                            ::
;::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
;

;   RULE 1::
(initea)
(prog (mode)
      loop
      (princ 'enter-mode)
      (setq mode (read))
      (cond ((eq mode 'EXIT)
             (done))
            ((eq mode 'DIAGRAM)
             (rule2 mode)
             (terpri)
             (print (list '----------TRANSLATING----------))
             (terpri)
             (rule3 'TRANSLATE))
            (t)
       )
      (go loop)
 )
```

APPENDIX 5

AN EXAMPLE DESIGN OF A MINIMAL DATABASE USING RULES BASED ON THE EXPERT ASSISTANT

```
XLISP version 1.5, Copyright (c) 1985, by David Betz
; loading "ea.lsp"
ENTER-MODE> diagram
ENTER-DATABASE> db1
ENTER-DBMS-ARCHITECTURE> network
ENTER-ENTITY> emp
ENTER-ENTITY-TYPE> e
(-------- FOR THE EMP ENTITY ---------)
INPUT-ATTRIBUTE-VALUE-IDENTIFIER> ssn
INPUT-ATTRIBUTE-TYPE> 1:1
ENTER-ATTRIBUTE-VALUE> 123456789
ENTER-UNIQUE-RELATION-IDENTIFIER> is-affiliated-with
ENTER-RELATION-TYPE> 1:N
ENTER-RELATION-DEPENDENCY> n
ENTER-SECOND-ENTITY> dept
ENTER-ENTITY-TYPE> e
(-------- FOR THE DEPT ENTITY ---------)
INPUT-ATTRIBUTE-VALUE-IDENTIFIER> dept-no
INPUT-ATTRIBUTE-TYPE> 1:1
ENTER-ATTRIBUTE-VALUE> 556
ENTER-ENTITY> emp
(DUP-ENTITY ENTER NEW RELATION)
ENTER-UNIQUE-RELATION-IDENTIFIER> manages
ENTER-RELATION-TYPE> 1:1
ENTER-RELATION-DEPENDENCY> n
ENTER-SECOND-ENTITY> proj
ENTER-ENTITY-TYPE> e
(-------- FOR THE PROJ ENTITY ---------)
INPUT-ATTRIBUTE-VALUE-IDENTIFIER> proj-name
INPUT-ATTRIBUTE-TYPE> 1:N
ENTER-ATTRIBUTE-VALUE> accting
ENTER-ATTRIBUTE-VALUE> payroll
ENTER-ATTRIBUTE-VALUE> end
ENTER-ENTITY> end

(----------TRANSLATING----------)


(----------PROCESSING DATA-DICTIONARY----------)
```

```
ENTER-MODE> exit
DATABASE-RECORD-LIST
--->DB1
--->((MEMBER-RECORD7 DEPT-NO 556)
     (OWNER-RECORD6 SSN 123456789)
     (MEMBER-RECORD4 PROJ-NAME ACCTING)
     (MEMBER-RECORD3 PROJ-NAME PAYROLL)
     (OWNER-RECORD2 SSN 123456789))


DATA-DICTIONARY-LIST
--->((DB1
     ((MEMBER-RECORD7 DEPT-NO 556)
      (OWNER-RECORD6 SSN 123456789)
      (MEMBER-RECORD4 PROJ-NAME ACCTING)
      (MEMBER-RECORD3 PROJ-NAME PAYROLL)
      (OWNER-RECORD2 SSN 123456789)))
     (OWNER-RECORD2 SSN MEMBER-RECORD3 PROJ-NAME
      MEMBER-RECORD4 PROJ-NAME OWNER-RECORD6 SSN
      MEMBER-RECORD7 DEPT-NO)
     (SSN 123456789 DEPT-NO 556 PROJ-NAME ACCTING
      PROJ-NAME PAYROLL))


FIELD-ATTRIBUTE-LIST
--->SSN
--->123456789
--->DEPT-NO
--->556
--->PROJ-NAME
--->ACCTING
--->PROJ-NAME
--->PAYROLL


DATA-STRUCTURE-LIST
--->(SET1)
--->(SET5)


MEMBER-RECORD-LIST
--->(DEPT SET5)
--->(PROJ SET1)
--->(PROJ SET1)


OWNER-RECORD-LIST
--->(EMP SET5)
--->(EMP SET1)
```

```
RECORD-FIELD-LIST
--->OWNER-RECORD2
--->SSN
--->MEMBER-RECORD3
--->PROJ-NAME
--->MEMBER-RECORD4
--->PROJ-NAME
--->OWNER-RECORD6
--->SSN
--->MEMBER-RECORD7
--->DEPT-NO


ATTRIBUTE-LIST
--->(PROJ PROJ-NAME 1:N PAYROLL)
--->(PROJ PROJ-NAME 1:N ACCTING)
--->(DEPT DEPT-NO 1:1 556)
--->(EMP SSN 1:1 123456789)


RELATION-LIST
--->(EMP PROJ MANAGES 1:N N)
--->(EMP DEPT IS-AFFILIATED-WITH 1:N N)


DATABASE-LIST
--->(DB1 NETWORK
     ((EMP SET5) (EMP SET1))
     ((DEPT SET5) (PROJ SET1) (PROJ SET1))
     ((MEMBER-RECORD7 DEPT-NO 556)
      (OWNER-RECORD6 SSN 123456789)
      (MEMBER-RECORD4 PROJ-NAME ACCTING)
      (MEMBER-RECORD3 PROJ-NAME PAYROLL)
      (OWNER-RECORD2 SSN 123456789))
     ((PROJ PROJ-NAME 1:N PAYROLL)
      (PROJ PROJ-NAME 1:N ACCTING)
      (DEPT DEPT-NO 1:1 556)
      (EMP SSN 1:1 123456789))
     ((SET1) (SET5))
     ((PROJ POINTER12 EMP)
      (PROJ POINTER11 PROJ)
      (DEPT POINTER10 EMP)
      (EMP POINTER9 PROJ)
      (EMP POINTER8 DEPT)))
```

```
POINTER-LIST
--->(PROJ POINTER12 EMP)
--->(PROJ POINTER11 PROJ)
--->(DEPT POINTER10 EMP)
--->(EMP POINTER9 PROJ)
--->(EMP POINTER8 DEPT)


ENTITY-LIST
--->(PROJ E)
--->(DEPT E)
--->(EMP E)


RECORD-LIST
--->(MEMBER-RECORD7 DEPT-NO 556)
--->(OWNER-RECORD6 SSN 123456789)
--->(MEMBER-RECORD4 PROJ-NAME ACCTING)
--->(MEMBER-RECORD3 PROJ-NAME PAYROLL)
--->(OWNER-RECORD2 SSN 123456789)
```

Expert Assistance for Database Design

by

Roger Allen Vasconcells

B.S., Kansas State University, 1980

AN ABSTRACT OF A MASTER'S THESIS

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1987

ABSTRACT

The design phase of a database management system forms the foundation of its usefulness. Various complicsted tools have been developed that assist the database designer with this process. The simplistic Entity-Relationship Approach to database design has received much interest and use.

This thesis presents a formalism that would provide assistance to the database designer. The Expert Assistsnt, based on protocols defined by the EMYCIN expert system construction tool, allows the knowledge of previous users of the Entity-Relationship Approach to datsbase design to be stored and accessed. By monitoring the progress of s designer using this system, the Expert Assistant will provide assistance with decisions and insure that s visble dstabase management system is developed.