1-1-1978

Database design: A practical methodology.

Kerry Nemovicher

DATABASE DESIGN: A PRACTICAL METHODOLOGY

by

Kerry Nemovicher

A Thesis

Presented to the Graduate Committee

of Lehigh University

in Candidacy for the Degree of

Master of Science

in

Industrial Engineering

Lehigh University

1978

This thesis is accepted and approved in partial fulfillment of the requirements for the degree of Master of Science.

Aug 21, 1978
(date)

Professor in Charge

Chairman of Department

ACKNOWLEDGMENTS

Although only one name appears upon the title page, this thesis has come into being through the combined efforts of a number of individuals. The author wishes to thank Dr. Ben L. Wechsler for his persistent encouragement and careful scrutiny of the ideas presented herein. His critique has eliminated much of the fuzzy thinking inherent in the original conceptualizations and drafts. The author also wishes to acknowledge the help of Dr. Larry E. Long, Mr. Dia Ali, Dr. Joseph Nemovicher, and Mrs. Rae Nemovicher for their careful reading of the successive drafts of this thesis and their roles as sounding boards for the ideas developed. The patience and understanding shown by Marie Young and Joanne Whitney in typing (and retyping) this text has been more than this author deserves. Finally a special thanks must go to Mrs. Marcia Berkow, the sine qua non of the final form of the thesis. Her skills as an editor, proof reader, sounding board and typist have rendered the formidable task of putting this thesis into final form, a genuinely pleasant undertaking.

TABLE OF CONTENTS

LIST OF FIGURES

PAGE

PAGE

PAGE

ABSTRACT

A PRACTICAL DATABASE DESIGN METHODOLOGY

This thesis presents a practical, implementable methodology for Database logical design. In the rapidly expanding world of Database users the quest for viable and efficient logical designs is one of the most difficult and elusive of tasks. As the uses of Database technology grow, not only in number but also in complexity, the need for a design methodology becomes daily more immediate.

This need has been recognized throughout the industry and in recent years a number of Database design methodologies have appeared on the market. Unfortunately these methodologies have proved less than satisfactory. In general they have tended to be overly concerned with machine optimization (to the detriment of the logical design), and/or are so theoretically abstract and esoteric as to be beyond the ken of the potential users -- the practitioners in the field.

The aim of this thesis is to remedy the situation by presenting a practical, logical design oriented methodology. The approach is an intuitive one based upon the understanding that:

- a Database logical design deals in generic concepts (or data item types, herein referred

to as eide) and not in particular items of data

- a Database exists in order to supply information, the components of which are data and the relationships between data
- the data which comprise information can be viewed as functioning either as objects (the principle "subjects of concern") or as attributes (modifiers or descriptors of the principle objects)
- a Database must be able to accommodate the information needs of all its users
- an optimal logical design is one in which the information needs of all of the users are readily discernible.

The methodology can be divided into two phases:

1. Phase I - identification of the information to be modeled
2. Phase II - construction of the information model (the conceptual design).

The logical design of a Database can be thought of as the conceptual structure of the Database's information capacity, an information map. It contains the relationships between the generic concepts (not the actual data) which constitute the analyst's conceptions

of the systems to be supported. Thus the first phase in the logical design process is the traditional task of the systems analyst -- understanding (conceptualization) of the systems to be supported.

The second phase is the task of converting the analyst's understanding of each of the separate users' views of the system into an integrated whole -- the conceptual design of the Database. This is accomplished almost mechanically, first by mapping each user's view onto an object/attribute matrix (a table listing "objects" on one axis, "attributes" on the other and their relationship at the intersection) and then systematically constructing records and sets (relationships between records) from the information contained in the matrix.

The process is as effective as it is simple. The resulting logical designs (conceptual schemata) are both logically sound and efficient. They compare favorably with the best of the esoteric and machine oriented designs, yet are far less complex. Most important, the methodology is comprehensible and implementable. A Database is first and foremost a practical, pragmatic tool, a tool meant to be used; so too this Database design methodology is a practical pragmatic tool - meant to be used.

CHAPTER ONE

STATEMENT OF OBJECTIVE AND PROBLEM SCENARIO

I. Background

There is no doubt today that the Data Processing (DP) industry has accepted the concept of the "Database". Software vendors have invested considerable sums of money in Database development, and a plethora of Database packages have inundated the market (Wiederhold, 1977). The professional journals abound with articles about various aspects of Database technology and the Association for Computing Machinery (ACM) now publishes a quarterly devoted solely to Database related topics ("ACM Transactions on Database Systems"). Many of the larger DP centers already have operational Database based systems and the trend certainly seems to point toward an exponentially expanding Database user population. As James Martin indicates, "The development of corporate data bases will be one of the most important data-processing activities for many years to come."(Martin, 1977, p. 2)

Yet, despite its apparent popularity, the development and implementation of a Database system is not a simple matter. For the most part it is a far more difficult undertaking than the development

and implementation of traditionally structured systems. The software (as will be discussed below) and the hardware problems are significantly more complex than those encountered in non-Database systems, and as with most "new concepts", the human inertia-to-change can be a serious obstacle.

Even if one restricts one's view to just the software, there are many facets to the problem of Database design. Among other things the designer must concern himself with efficient storage utilization (balanced against rapid information retrieval), security (insuring both privacy and protection of information in the event of machine failure), data integrity (the "absence of inconsistent data"), and a sound "logical design" (the grouping of data items into records and the definition of the interrelationships between those records). It is this last aspect, Database logical design, which is the subject of this thesis.

In traditional (non-Database) systems, a file is designed around a specific application. Thus the manner in which the data are stored (and the interrelationships between data items) can be optimally designed to accommodate the information needs of the specific application. A Database system on the other

hand, must serve many applications and yet use a single data storage scheme. This scheme must be flexible enough to allow each application to view the data as if an "application-oriented" file existed. It must accommodate all the necessary data items and all the necessary interrelationships between those data items.

To create and support a flexible storage scheme it is necessary to know the nature of all the constituent data item types and the nature of all their interrelationships. This knowledge is recorded in an "information map", which, in Database parlance, is termed a Schema. A schema contains (among other things) a list of all the data item types which appear in a Database, how they are physically organized, and how they are logically connected. A good schema must not only accommodate all the "current" application views of the data, but must try to anticipate what views will be needed in the future. It is easy to imagine how, with applications of any significant complexity, a poorly planned schema could grow to become extremely tangled and unwieldy.

One of the major problems, therefore, of Database design is the design of the schema. It is a task fraught with complexities, one which can appear

overwhelming in its enormity. In most circumstances the design of the schema is critical to the overall Database design and a poorly designed schema will almost certainly doom a Database.

II. Statement of the Problem

The problem is how to design a viable Database schema. When faced with the task, where does an analyst begin? Given both the importance of the job and the newness of the problem (the lack of experienced expertise) a need exists for a tool, a technique or methodology which can be used as a guide to schema design.

This need has begun to be recognized and Database design books and articles are proliferating (Chapter Two consists of a brief survey of some of the more popular design directions). Unfortunately there are serious drawbacks to the methodologies which have appeared to date (Chapter Three contains a critique of the methodologies discussed in Chapter Two). For the most part the proposed schema design methodologies rely too heavily upon the user's knowledge of sophisticated mathematics and/or they place too much importance upon the machine optimization aspects of Database design.

Thus, the need for a workable and practical schema design methodology still exists. It is a vacuum which must be filled. With the growing use of Database

technology and the growing desire to "go Database", the need for viable schemata (and hence the need for a technique to design viable schemata) becomes daily more immediate. It is a problem which demands a solution.

III. Statement of the Objective

It is the aim of this thesis to provide a simple and concise Database schema design methodology. The methodology, above all else, must be a practical one. It must be applicable to real world situations and implementable by persons possessing the skills and knowledge of today's DP professionals.

To design a methodology, no matter how powerful, which is incomprehensible to the people who must work with it is to defeat the very purpose of the methodology.

IV. Approach to the Problem

A Database designer must contend with many problems. If the tools which are meant as aids and guides are also complex and challenging, then the actual basic problems themselves are magnified. The methodology presented herein is an attempt to provide a positive tool for the Database designer, a tool which will lessen the load and free the analyst for the basic "analytic" problems of system design.

As is indicated above, there have recently been many Database design and schema design techniques appearing in books and articles, and although, as also indicated above, none of these can be regarded as definitive, the examination and understanding of these proposed methodologies is the obvious first step in an attempt to derive an effective schema design methodology. Chapter Two begins this process by providing a brief presentation of some of the more popular design directions. Chapter Three deepens the understanding by critically examining each of the design directions and attempting thereby to determine the most desirable design methodology characteristics. By understanding both the successes and shortcomings of the first ventures one can gain a knowledge of the problems to be overcome and the ideas which have proven fruitful.

The final step is the synthesis, the combination of the author's own field experience with the understanding acquired in the examination of the ideas of others. This is presented in Chapter Four. It is the exposition of a schema design methodology which is both flexible and simple to use. Although the presentation is perhaps far more detailed and theoretical than would be necessary for a practitioner, the methodology itself is straight forward and readily

automatable. It is neither an esoteric, (mathematical or symbolic logic based) technique, nor one that is dominated by machine optimization considerations. It is, in short, a practical schema design methodology.

CHAPTER TWO

CURRENT DESIGN METHODOLOGIES

I. Introduction

When dealing with an entity as large and complex as a Database it is almost inevitable that various different aspects of it will come to be regarded as its most important quality. One is reminded of the story of the blind men and the elephant; where each blind man, having encountered a different part of the elephant's anatomy (and of course unable to see the whole creature) came to a unique conceptualization of the nature of elephants. This would certainly seem to be the situation of the DP world and its encounter with Database. Published Database design techniques run in such varied directions (in accordance with each author's own understanding of what is "most important") that it is difficult to believe that all are concerned with the same end product.

In this chapter, some of the major Database design techniques will be reviewed. Insofar as the subject of this thesis is Schema design, this is the "important quality" which this author will emphasize. Whenever applicable, however, an attempt will be made to show how schema design fits into the total perspective

of a particular Database design scheme.

It will be remembered that the schema must describe the three elemental aspects of a Database: the data item types to be stored, the grouping of data item types into records, and the grouping of records into sets. A successful schema must do this in a way which will accommodate all current and foreseen uses of the data. The manner of this accommodation must be as efficient and straightforward as possible, keeping to a controlled minimum the need for data redundancy and long involved sequential searches.

As has been indicated, this is no mean task, growing exponentially with the size and complexity of the system. Judging from the literature, however, the early Database theorists apparently did not realize the magnitude of schema design. Today, with the wisdom of hindsight and experience, the problem is being given more attention. In the following pages the reader will find a sampling of some of the basic design directions, running the gamut from simple intuitive design up to the most recent attempts at "automated" Database design.

II. The Artistic Intuitive Approach

A direct carryover from traditional system

design practices is the intuitive approach to Database design. In traditional design, typically the analyst collects all of the pertinent information about "who needs what in which report" and then intuitively sees what data items are needed, how the data items are to be grouped into records and which records are to belong to which files. Occasionally the analyst is required to perform a few calculations in connection with the use of a sophisticated data storage technique (such as indexed sequential), but for the most part an experienced analyst intuitively knows what to do. Understandably a system design, like a computer program, often bears the unmistakable artistic stamp of the designer.

Many analysts assume that this same basic intuitive process will work with Database systems. They propose that to design a Database one must understand the physical mechanisms and organization of Database data storage techniques as well as the traditional designers understand the hardware and software devices of traditional technology. Armed with this understanding "of the tools" and a sufficient knowledge of the applications to be supported, an analyst should be able to intuit a viable Database design.

III. The Machine Efficiency Oriented Approach

The early Database packages seriously strained the capabilities of their host machines. Consequently many of the initial Database design and implementation efforts (as well as the software itself) centered around the efficient use of the machine. "Although the original data base packages were conceived in the glow of program independence, they were delivered by the practitioners of machine/core optimization." (Wood & Chamberlain, Feb. 6, 1978, p. 26). This concern with machine efficiency is naturally reflected in the design techniques of those who use machine oriented packages and by some who have been simply swept along in the wake of such packages. Two such machine-oriented techniques are briefly described below. The first was designed for use with IBM's IMS (Information Management System). The second is a computer automated methodology for "Optimal Data Base Schema Design" of CODASYL DBTG type Databases.

A. Service Analysis

Service Analysis is a Database design methodology marketed by Advanced Systems Incorporated (ASI). Although aspects of Service Analysis are adaptable to many Database Management Systems, it is primarily intended for use with IBM's IMS.

Consisting of thirteen design tasks (see Appendix B) the critical design criteria are: data item frequency of access, data item size (number of characters), and data item number of potential occurrences. The design goal is to arrange the data item types (or "fields") in such a manner as to render those which are most frequently accessed most easily accessible. One would never, for example, group a large, rarely accessed data item together with a highly used one since each access of the volatile data item would have to carry with it the dead weight of its unused companion.

Although in the Service Analysis approach the physical structure orientation dominates the logical relationships, it is understood that the logical (or generic) relationships must also play a part. The interaction, however, between physical and logical is a difficult one to define. As Katzen states: "the synthesis of effective logical and physical data structures is more of an art than it is a science at this point in the evolution of data base technology" (Katzen, 1975, p. 298). Service Analysis seems to want to design the physical structure (the design methodology for which is very detailed) and then let the designer intuit the superimposition of the logical

upon the physical.

Where, however, do the definitions for the requisite logical relationships come from? Service Analysis leaves this question for the designer. It is this author's personal understanding that ASI left the problem for "artistic intuition". Others have treated it as a separate problem and attempted to develop a logical design methodology divorced from the physical considerations. (This thesis is one such effort). Later in this chapter, some of those "purely logical" approaches will be examined.

B. An Automated Optimizer for DBTG Type Schema Design

In a document worthy of Laputa,[1] Michael F. Mitoma presented "a methodology that will automate and optimize the design of DBTG database structures for specified applications" (1975, p. i). Of course, as Mr. Mitoma himself points out "there is no widely accepted definition of the somewhat nebulous concept of 'optimality' as it applies to data base or file design" (p. 29). He defines "the optimal data base to be the one which supports the required processing with

---

[1]The island floating in the sky which was discovered by Captain Lemuel Gulliver. Captain Gulliver later commented that: "neither Prince nor People appeared to be curious in any Part of Knowledge, except Mathematicks and Musick,".

the lowest logical record access frequency, subject to a number of feasibility and storage bound constraints" (p. 29).

Necessary input to this optimizer consists of a complete description of the data items, the data base relations (which data items are related to which), the run units of the system (with regard to how each run unit is to relate to each data item) and so on.

The design steps are as follows:

1) construct a mathematical model of the "data management problem" (i.e. the data items, the data base relations, the run units, and the schema storage space);

2) construct a mathematical model of the "data base schema," consisting of the number of occurrences of each data item, a description of legal data aggregates, records and possible sets;

3) manipulate these two models into what is called "the optimization module," (i.e. an Integer Program formulation);

4) construct the Integer Program itself from the formulation;

5) run the Integer Program to produce the optimal schema.

The basic idea behind the technique is to examine every feasible configuration and then compute

for each one the access paths necessary for each data item type. A data item type may, after all be accessed in its own right, or as related to another data item type (as part of a relationship). The integer program, taking into account the access frequency expectations for each data item type, determines the optimal schema configuration to be that which has the shortest "average" access paths.

Despite the author's statement that his methodology "considers logical design only . . . not the physical organization of the data base," (p. 15) it falls naturally into the category of "machines optimization" orientation. Mr. Mitoma clearly and concisely expresses the philosophy of that orientation in the following:

> Because a record is the basic unit of access in DBTG systems, we measure the efficiency of a schema and its corresponding data base with respect to a given data management problem in terms of the number of record accesses which are required to accomplish the processing described in the run units of that data management problem. In particular, the optimal data base is the one on which the required processing can be performed with the fewest total number of record accesses, subject to the constraint that the size of the data base must not exceed the maximum capacity of the schema storage space. The optimal schema is then the schema that describes this data base (p. 102).

IV. The Functional Approach

One of the more successful concepts which has

taken hold in recent years has been the idea of "top-down" design. Useful in both systems analysis and in programming, the top-down design approach initially analyzes a problem at the very general major function level and then gradually divides each major function into successively finer sub-functions.

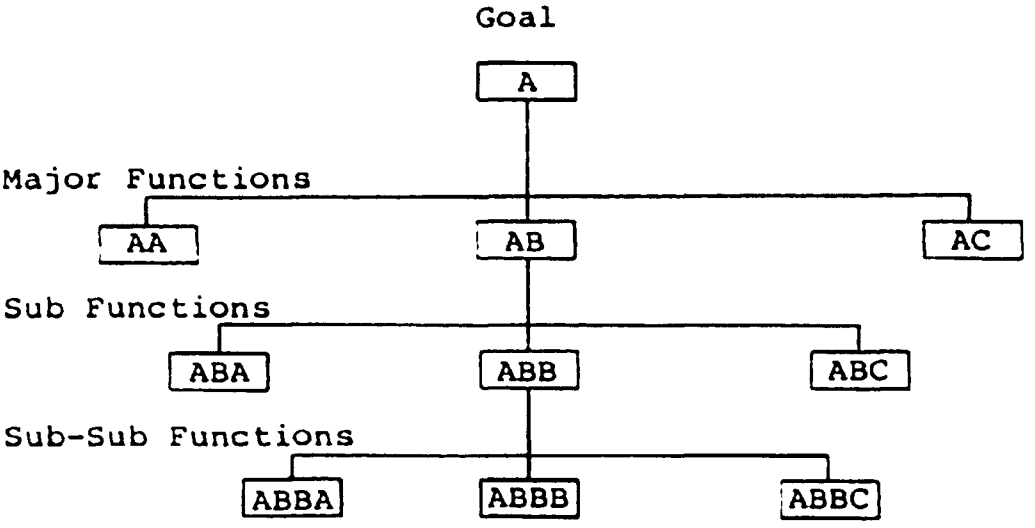


Figure 2.1 Functional Level Tree

Recognizing the value of this approach, analysts have applied it to Database design. Typically it is referred to as being the "Functional Approach" to Database design. Its advocates generally stress the pan-application aspects of Databases and like John K. Lyon define Database as "The total data resources of an organization, considered as an entity" (1976, p. 2).

In an organization which already has a considerable tradition of departmentalized information, the use of the functional approach is not quite as obvious as it may seem. In their effort to "go Data-base" many DP installations have simply tried to link together their traditional parochial files. Others have tried the top-down approach strictly along company departmental lines (i.e. viewing each department as a major functional unit).

Such efforts rarely proved successful. Instead, argue supporters of top-down analysis, a true functional analysis must take place. "Top-down design will work for data bases as well as it does for programs if the analysis is done by function rather than by department" write Thomas R. Finneran and J. Shirley Henry in Datamation. "If the designer divides the 'big picture' into business functions, rather than organizational entities, he can logically segment the organization, avoiding the problems inherent in departmental analysis" (Nov. 1977, p. 99).

The process then of functional design is as follows:

1. identify the major enterprise (or system functions);
2. construct a "functional tree" by dividing

each function into its constituent sub-functions until the elementary level functions are reached;

3. identify the data elements necessary to support each elementary function;
4. identify and combine identical branches and branch segments;
5. combine "similar" segments and branches through the use of "more generalized" definitions;
6. form records (segments) out of the data item "leaves" of each elementary level branch;
7. "infer" sets "from their common key elements" (i.e. records which contain a common data item are related in an obvious way).

The philosophy here is clear.

> If a Database is to truly exist, it must be designed with the understanding that it is not just an on-line file, but, rather, that it is, first and foremost, a model of the business. . . . The Database administrator /designer/ has an obligation to the company to insure that the Database is defined in terms of the essential basic entities of the enterprise and not in terms of the way things are done now, nor in terms of a user's idea of an implementation. The process of Database design is one of determining those areas that are important and significant to the enterprise and its environment" (Lyon, 1976, p. 19).

V. Information Oriented Approaches

In recent years some analysts have attempted to view the problem of Database design in the light of what must be regarded as the essential aspect of a Database: information. Proponents of this position contend that a Database should be a repository of retrievable information, not just an amorphous pool of data. The Database design should therefore emphasize the information capacity required of the Database rather than machine efficiency considerations.

Much thought has been invested in trying to determine the basic elements of information and how best to describe them. The initial division of information into entities (objects) and relationships between entities is almost universally accepted. The issue becomes clouded, however, with the introduction of auxiliary concepts such as properties, attributes, constellations, upper and lower conceptual domains, predication structures, et cetera. An interested reader soon finds himself traveling in the realms of semantics, symbolic logic, set theory, relational calculus, relational algebra, and an occasional sojourn into classical philosophy.[2] How do such esoteric topics pertain

---

[2]Bo Sundgren even cites Heraclitus (the obscure?) in his discussion of "existence".

to Database design? Here are some examples.

A. An Infological Approach to Database Design

Infological, a relatively recently coined word, simply means information oriented. The proponents of the infological approach claim that the users of a database are information oriented, hence, information oriented (infological) implies user oriented.

> The distinction between information and data is essential to the infological theory of data bases. . . . Very briefly, data are the materialization, the representation of information, whereas information is equivalent to knowledge and has to do with the semantic aspect, the meaning of data.
>
> By distinguishing between information and data we may similarly distinguish between infological, or user oriented, and datalogical, or computer oriented, data base design problems (Sundgren, 1975, p. 2).

The infological approach to Database design requires the designer to analyze the system in terms of the information which the system is required to produce. Information is viewed as being a reflection of "reality". The mapping of "reality onto a Database"[3] (reality⟶Database) takes place as follows:

1. The reality⟶subject matter model

---

[3]For this discussion Bo Sundgren defines "a data base as a permanently maintained digital data model of a slice of reality" (Sundgren, 1975, p. 18).

submapping, established by the abstraction process;

2. The subject matter model ⟶ infological model, established by the specification process;

3. The infological model ⟶ datalogical model submapping, established by the design process;

4. The datalogical model ⟶ data base submapping, established by the implementation process and maintained by the operation process (1975, p. 18).

In less technical terms this means that:

1. the analyst views "reality" and subjectively forms an understanding of what he sees;

2. the subjective understanding is modeled using some sort of formalized modeling technique (in this case, an "infological model");

3. the formalized information model is translated into a working machine model (a working schema);

4. the actual data for the Database is loaded and the Database is maintained.

The real substance of the infological approach is concerned with step #2, the construction of an infological model from the analyst's perceptions of the system. Unfortunately the terminology and the concepts of this technique are so esoteric and foreboding that it is impossible (within the limits of this discussion) to either delve into the requisite explanations or to translate the jargon into the vernacular. The interested reader is referred to Appendix B for an introductory

discussion of the subject (and to Bo Sundgren's book Theory of Data Bases for comprehensive coverage). This introduction and above cited quotations, however should be sufficient to convey the unique flavor of the info-logical approach to Database design.

B. The "Entity-Relationship Model"

Most of the schemata in use and being designed today contain "impurities", that is to say that they are

> not a "pure" representation of the real world. One of the reasons is that the database designer is restricted by the limited capabilities of the database management system. For example, the many-to-many relationships between entities are difficult to represent directly in some database systems. Another reason is that the user schema may contain some features related to the storage representation of the database. For instance, it may describe which record types can be directly accessed and how to access other record types. In addition, the user schema is usually designed to be efficient for a certain type of data processing operations. . . . Therefore, the user schema is usually not a direct representation of the real world. This makes the user schema difficult to understand and difficult to change (Chen, 1977b, p. 77).

What some analysts propose as a remedy for this situation is the creation of a super-schema (or meta-schema, enterprise schema, or conceptual schema). It is the task of the super-schema to accurately reflect the "real world" free from all pragmatic "state of the art" limitations. Such a super-schema would then be

hardware and software independent, conceptually easier and truly information oriented. Specific working schemata could easily be derived from a super-schema by applying the particular restrictions of a specific Database Management System.

One of a number of super-schema design methodologies is Peter Pin-Shan Chen's "Entity-Relationship model" (usually abbreviated as "E-R model"). In the E-R model terminology the super-schema is called the "enterprise schema" (supposedly because it accurately reflects the true enterprise view of information and not some view forced to fit the confines of a restrictive computer dictated model).

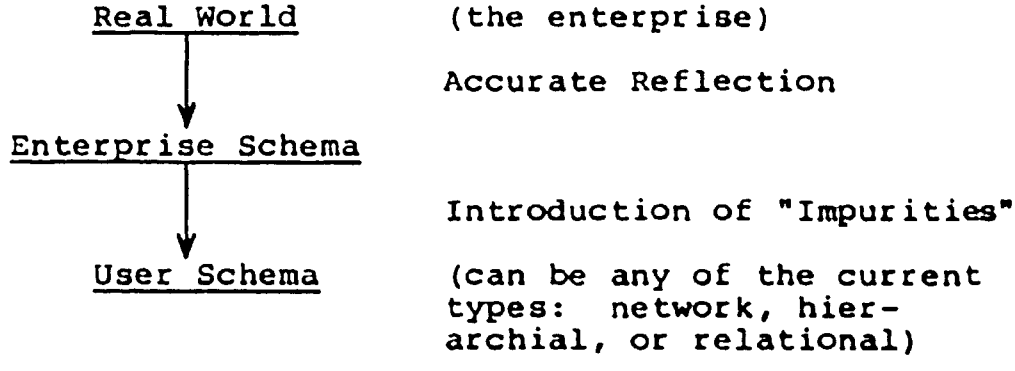


Figure 2.2 The Role of the Enterprise Schema

The E-R model utilizes five basic concepts. These are (Chen, 1977b, pp. 77-79):

1) entity sets - "An entity is a 'thing' which can be distinctly identified. An entity set is a group of

entities of the same type." In common terminology an entity would be called a datum (or a data item), and an entity set would be called a data item type (or generic name).

2) relationship sets - "Entities are related to each other." A specific department (an entity) may consist of a number of employees (entities). Between the department and each employee there is a relationship. The set of all the relationships between all the departments and all the employees is a relationship set. "A relationship set is a set of relationships of the same type." Another way to approach the definition of relationship sets is to note that relationships exist between entities and relationship sets exist between entity sets.

3) attributes - An attribute is that aspect of an entity or a relationship which can be assigned a value. Age, for example, can be an attribute of an employee because a value (number of years) can be assigned to it. In the E-R jargon an attribute is defined as "a mapping from an entity set (or relationship set) to a value set (or a group of value sets)."

4. value sets - As with entities and relationships, a "value set is a group of values of the same type." Color, number of years, name of locations, height, etc., all could be value sets.

5) conceptual domains (upper and lower) - Conceptual domains are concepts used to accommodate changes to the enterprise schema design (they are, however, beyond the scope of this brief presentation, but are covered in more detail in Appendix B).

These five basic concepts are the components of the E-R model; the media for the "enterprise" (or conceptual) schema. The E-R approach proposes that the design process begin with the construction of the E-R model. Using that model as a conceptual base, a working schema can then be produced.

The E-R model is meant to be a design tool, bridging the gap between the human understanding of an enterprise and how that enterprise must be modeled in the machine. It is claimed that the E-R model "adopts the more natural view that the real world consists of entities and relationships," and that it "incorporates some of the important semantic information about the real world" (Chen, 1976, pp. 9-10). It is a serious attempt to establish a comprehensible logical approach as the foundation for database design.

C. Semantic Predication Analysis

One of the methods chosen to convert human information into machine retrievable information is semantic predication analysis. It is used as the principle methodology of what is (unfortunately) called the Functional Model (totally unrelated to the functional approach to Database design described in section IV). It assumes that there must be at least a word picture of the enterprise (or system) available. Given the word picture, semantic analysis analyzes the semantic structure of the sentences to produce a Functional Model; which in turn can then be converted into a schema.

At the highest level (the information model level), "the Functional Model of data is viewed as a directed graph; its nodes represent sets and its arcs represent total functions. Nodes are either entity sets or value sets. Entity Sets may have any number of incoming or outgoing arcs; Value Sets may have only incoming arcs, because 'values' are the ultimate logical representation of information" (Sibley & Kerschberg, 1977, pp. 86-87).

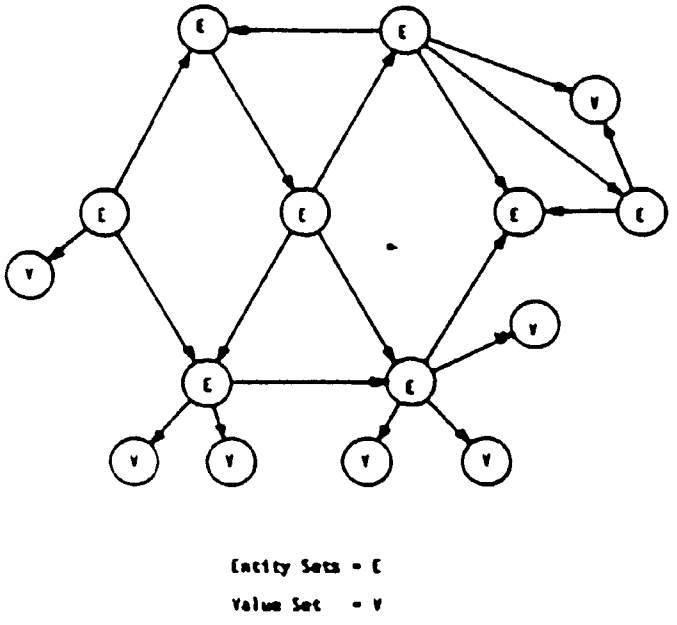


Figure 2.3 Functional model graph (Sibley and Kerschberg, 1977, p. 87)

> A predication represents a whole sentence; e.g., an assertion, a command, or a question; it may be decomposed into zero, one, or two arguments and a predicate. Arguments may themselves be predications. "Downgraded predications" may qualify arguments (the semantic equivalent of adjectival clauses) or may modify predicates (the semantic equivalent of adverbial clauses). The lowest semantic level consists of semantic features which serve as atomic semantic description units. Down-graded predications play the role of semantic features of the arguments or predicates that they qualify or modify (Sibley & Kerschberg, 1977, p. 87).

Consider the following example from Sibley and Kerschberg.

1) Statement: "Companies supply parts to departments in some volume."

The main predication structure (PN_1) is "companies supply parts". It consists of two arguments: "companies" (A_1) and "parts" (A_2), and of one predicate: "supply" (P_1). Note Figure 2.4. The arrow under "supply" indicates the direction of the relationship of PN_1. Both PN_2 and PN_3 are "down-graded" predications. PN_2 is the predication representing the indirect object. PN_3 represents the adverbial information.

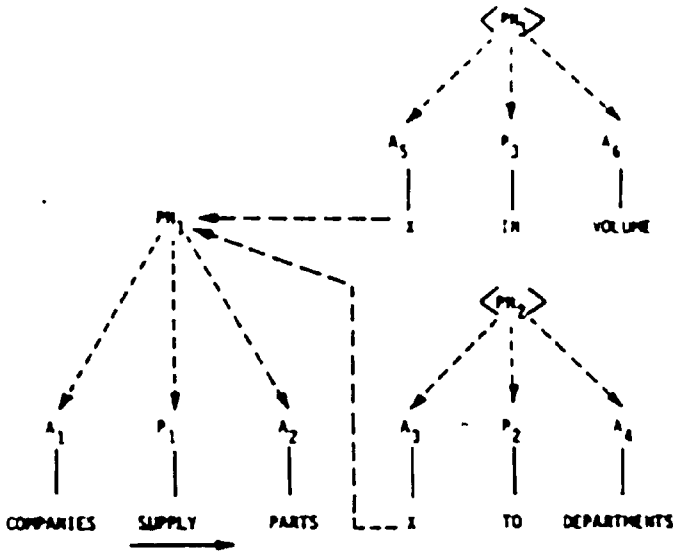


Figure 2.4 The predication structure for the sentence: "Companies supply parts to departments in some volume" (Sibley and Kerschberg 1977, p. 87)

The choice of the abstraction used to map predication structures to Functional Model data structures is part of data policy. As an example, the model might be restricted as follows:

- Semantic features map to functions whose range sets are value sets.
- Arguments corresponding to "real-world" entities map to named argument sets.
- Predications map to named predication sets, and the arcs pointing to arguments become named functions. Also the predicate and its arrow are attached to the predication set.

. Downgraded predications are represented by functions whose domain is the main predication set and range is either an argument set or a value set (Sibley & Kerschberg, 1977, p. 88).

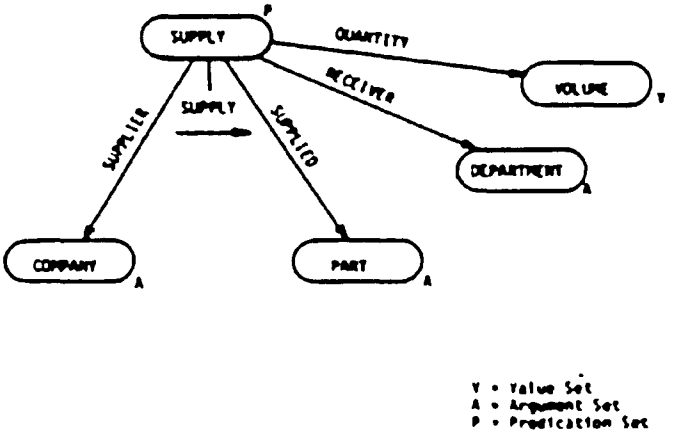


Figure 2.5 The functional model data structure (Sibley and Kerschberg, 1977, p. 88)

Figure 2.5 "depicts the Functional Model data structure based upon the predication structure of [Figure 2.4] . . . and the above abstraction rules" (Sibley & Kerschberg, 1977, p. 88).

Further abstractions are possible by choosing

a different set of abstraction rules. Sibley and Kerschberg maintain that by the appropriate choice of rules (Data Policy) one can eventually arrive at schemata of any of the three major practical types: network, hierarchy, relation.

VI. Canonical Schema Design

Does the information stored in a Database contain its own inherent structure? If it does, can this structure be used to derive an "optimal" schema? Some analysts have answered "yes" to both these questions and have produced what they refer to as being "canonical" schemata.

In the words of James Martin (1977, pp. 248-249):

> We will define a canonical schema as a model of data which represents the inherent structure of that data and hence is independent of individual applications of the data and also of the software or hardware mechanisms which are employed in representing and using the data.
>
> It is desirable to step away from the current software for a moment and ask the question, "Are there any inherent properties in the data which would lead to data items being grouped and groups being interconnected in a particular structure?" The design procedure described /below/. . . makes such structures clear. We refer to them as canonical structures. To be inherently stable, and be able to evolve naturally to meet the needs of new applications, a data base should have a canonical structure. This gives it the best chance of surviving future changes. It minimizes the risk of having to rewrite application programs because of data-base changes.

Canonical schemata are similar to the "information-oriented" models (discussed above in section V) in that they are not specific schemata for specific DBMS packages, but rather are once removed models of data structure from which working schemata can be derived.

> The canonical form of data. . . is independent of whether the data will eventually be represented by means of hierarchical, CODASYL, relational, or other structures. An additional step in deriving a workable schema is to convert the canonical form of the data into a structure that can be supported by whatever software is being used. This is a relatively straightforward step (Martin, 1977, p. 249).

The methodology for producing a canonical schema is not conceptually complex. It is so non-complex that it has, in part, been automated (Raver and Hubbard, 1977). The canonical design technique is an "incremental" one, based upon how each of the various applications "view" the data. Each user's view is added to (or can be deleted from) the canonical model in discrete steps, thus allowing for relatively simple model modification.

If one examines the canonical approach in light of the three basic schema design tasks (identifying data items, grouping data items into records and grouping records into sets) one finds that the canonical approach concerns itself with only the latter two tasks. It

assumes that the relevant data items have been identified for each user. This implies that either an existing set of applications can be called upon for this information or that some other function (such as the preparation of a Data Dictionary) has already performed the identification task.

The graphical tool of the canonical approach is the "bubble chart". A bubble chart is constructed by drawing each data item type as an elipse (or a circle) and by connecting related data items types with arrows. An arrow may be either single headed (for 1:1 relationships) or double headed (for 1:M relationships). An arrow may also be either uni-directional (indicating that only a relationship in one direction is relevant) or bi-directional (indicating that complementary relationships exist). Figure 2.6 contains an example of a bubble chart.

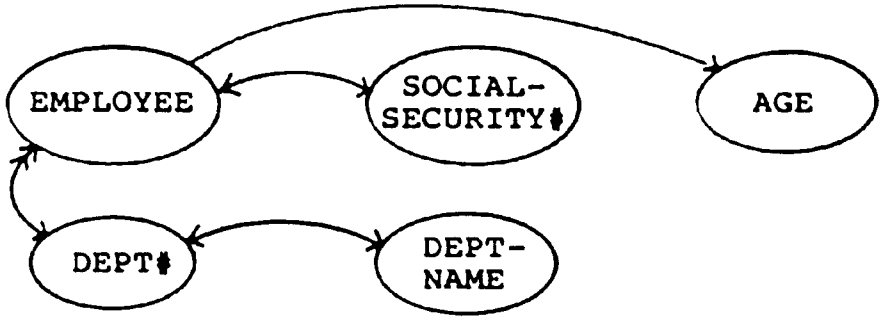


Figure 2.6 Bubble Chart Example

The canonical design process begins by representing one user's view (the information needs of one application) in bubble chart format. It then proceeds to incorporate additional users' views into the bubble chart until all the users' views have been taken into account. Redundant relationships (relationships which can be deduced from the existence of other, non-redundant relationships) are removed as they are discerned. The final bubble chart is then converted into a schema as follows:

All bubbles with single headed arrows leaving them are regarded as "prime keys" (i.e. the key field in a record). All bubbles which have only double-headed arrows leaving them become "secondary keys" (i.e. keys accessed through the use of a secondary index). All other bubbles are simply ("attribute") value fields. Figure 2.7 illustrates this process.

By no means an "ivory tower" product, the canonical design recognizes the exigencies of an operational DP center. It takes into account the physical problems of response time and machine utilization, but rather than treat the physical constraints as being of primary concern (in the manner of Service Analysis), it subordinates them to the more "essential" logical design. When compromises are necessary due to

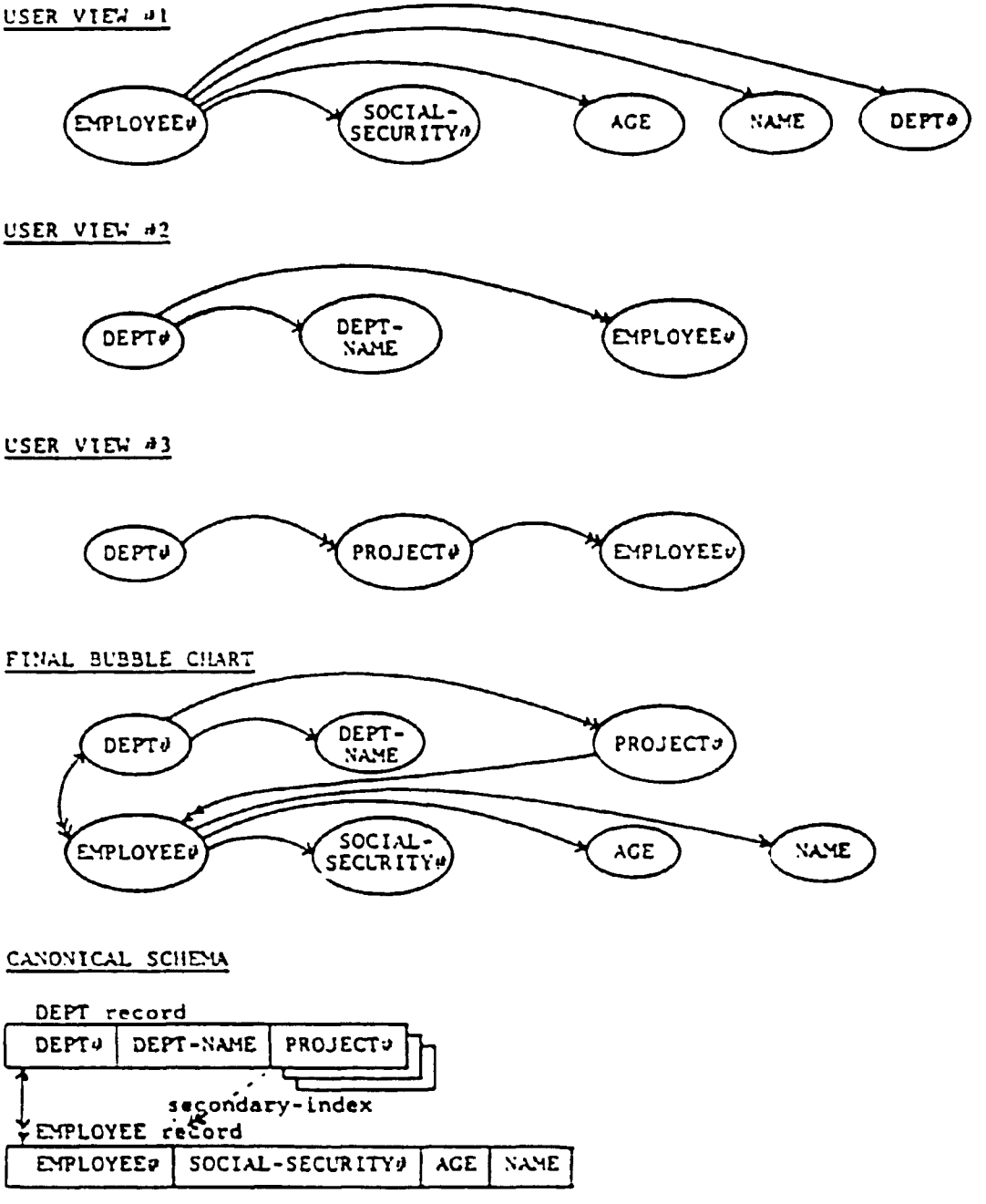


Figure 2.7 Canonical Design Process

physical constraints, the canonical approach accepts them for what they are: compromises due to limitations in the state of the art, not essential Database features.

VII. The Relational Approach

A movement which is rapidly gaining in popularity is the push for Relational Databases[4] and relational Database design techniques. For a discussion to do justice to Relational Database theory would require a vehicle of far greater scope than this thesis. To avoid the topic completely would be to ignore an area of ever increasing importance. The following discussion, of necessity brief, aims to convey only the basic philosophy of the relational approach and in no way pretends to be mathematically rigorous.

To quote C.J. Date paraphrasing E.F. Codd, the objectives for the relational approach are as follows (1977, p. 457):

1. To provide a high degree of data independence;
2. To provide a community view of the data of spartan simplicity, so that a wide variety of users in an enterprise (ranging from the most computer-naive to the most computer-sophisticated) can interact with a common model (while not prohibiting super-imposed user views for specialized purposes);

---

[4]To date, most relational database packages are still not past the experimental state.

3. To simplify the potentially formidable job of the database administrator;

4. To introduce a theoretical foundation (albeit modest) into database management (a field sadly lacking in solid principles and guidelines);

5. To merge the fact retrieval and file management fields in preparation for the addition at a later time of inferential services in the commercial world;

6. To lift database application programming to a new level--a level in which sets (and more specifically relations) are treated as operands instead of being processed element by element.

The relational approach is another of the "logical design approaches". It is concerned with the "user's view" of the data, not with physical representation.

> Codd's principles [the foundations of the relational approach] relate to the user's view of the data, or the logical description of the data. It is very important to note that they do not apply to its physical representation. There are many ways in which a Codd [relational] data base could be physically structured (Martin, 1976, p. 95).

The underlying principle of the relational approach stems from an understanding that any logical data structure, no matter how complex, can be broken down and represented as one or more table(s) of "relations". Consider the following simple example adapted from C.J. Date (1977). The data structure in Figure 2.8 is a typical hierarchy. Each SUPPLIER

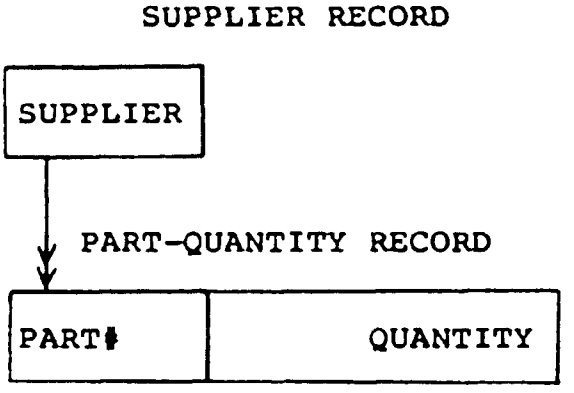


Figure 2.8 Simple Data Hierarchy

can supply multiple parts (PART#), each of which can be ordered in a given quantity. At any specific time the actual data being stored might consist of any combination of values. Such a possible set of values is given in table format in Figure 2.9. Although listed in a table, the structure is really still that of a hierarchy. SUPPLIER can be thought of as the key (the

SUPPLIER	PART-QUANTITY	
	PART#	QUANTITY
S1	P1	300
	P2	200
	P3	400
	P4	200
	P5	100
	P6	100
S2	P1	300
	P2	400
S3	P2	200
	P4	300
	P5	400

Table 2.9 Data in Tabular Format

root) of which PART-QUANTITY is a function. Likewise within PART-QUANTITY, PART# can be thought of as the key (and QUANTITY the dependent item). In order for Figure 2.9 to be considered a "table" in the relational sense it must undergo a process called Normalization. Figure 2.10 shows the same information as Figure 2.9 after normalization.

SUPPLIER	PART#	QUANTITY
S1	P1	300
S1	P2	200
S1	P3	400
S1	P4	200
S1	P5	100
S1	P6	100
S2	P1	300
S2	P2	400
S3	P2	200
S3	P4	300
S3	P5	400

Figure 2.10 A "Relation"

Such a normalized table of relationships is called a "relation". It is the basis for the relational approach. To qualify as a relation a table must meet a number of standards. According to James Martin the properties of a relation are (1976, p. 96):

1. Each entry in a table represents one data item; there are no repeating groups.
2. They are column-homogeneous; that is, in any column all items are of the same kind.

3. Each column (called a domain) is assigned a distinct name.

4. All rows (called tuples) are distinct; duplicate rows are not allowed.

5. Both the rows and the columns can be viewed in any sequence at any time without affecting either the information content or the semantics of the function using the table.

A relation, such as the one in Figure 2.10, is usually expressed as follows: SPQ(SUPPLIER, PART#, QUANTITY): where SPQ is the name of the relation and SUPPLIER, PART# and QUANTITY are its component domains (data item types).

A relational Database schema consists of a set of basic relations. Relations can be manipulated (through the use of relational algebra and relational calculus) to create various combinations of domains. Each new combination of domains is in effect a new relation. Thus any relation necessary to satisfy a user's view can be obtained at run time through the use of the relational algebra and relational calculus operations.

The basic relations, those which are to be Database resident and do not need to be created at run time, are what other methodologies term logical records. In order to be appropriate for use in a relational schema, these records must be in fourth normal form (4NF). A discussion of the complete normalization process which produces 4NF relations (records) is beyond

the scope of this Chapter, but can be found in Appendix B. In simple, non-mathematical terms, a 4NF relation (or record) consists of one key domain (an identifier or object) and unique attribute domains (i.e. attributes which are used solely as attributes of the key domain). The principle effort in relational schema design is the determination of the basic 4NF relations.

Based as it is upon a firm mathematical base, the relational approach is a favorite of the academic world. Unfortunately it suffers from a jargon filled vocabulary of confusion. Talented authors such as James Martin and C.J. Date are doing much to cut through the jargon and present the relational approach in a comprehensible manner, but at the moment (and in the near foreseeable future) the relational approach remains in the domain of the theoreticians.

VIII. Other Approaches and Combinations

There are many other approaches to Database design. There is another so-called "functional approach" (Gerritsen, 1975) which requires the designer to input each user's view expressed as a hierarchy statement (using a language called HI-IQ) into a software package called DESIGNER. DESIGNER then analyzes the total hierarchy picture and produces a schema network

in DBTG compatible format.

There is an automated package for estimating "total storage costs and average access time of several file organizations, given a specific data base, query characterization and device-related specifications. Based on these estimates, an appropriate file structure may be selected for the specific situation." (Cardenas, 1973)

In a similar vein, a group working for the U.S. Navy (Nunamaker, Konsynski, Ho, and Singer, 1976) have devised a system for "computer-aided analysis for the design and development of" computer based systems. Their system contains "four major components: Problem Statement Language, Problem Statement Analyzer, Generator of Alternative Designs, and Performance Evaluator." The Generator of Alternative Designs generates among other things "alternative specifications for program modules and logical database structures".

This list could easily continue and engulf many pages; as could the list of methodologies which make use of a combination of techniques. The combinations are mostly found in industry, the world of practical applications, where results are generally more important than strict adherence to any specific set of theoretical ideology. Combinations such as Service Analysis

interwoven with Raver and Hubbard's automated canonical schema design would probably prove very effective since each addresses a different (yet dovetailing) aspect of the total design.

The direction of the future is probably towards more and more automated design techniques. In addition to those already mentioned for established DBMS packages there is even some movement to automate the experimental ones as well [e.g. Philip A. Bernstein's package for "synthesizing third normal form relations from functional dependencies" (1976)7. The basis for automation exists; the design of a Database is intuitively a logical and orderly process. Yet before automation takes place, one should be sure that the methodology (upon which one bases the automation model) is the one "true" enough to warrant automation.

CHAPTER THREE

ANALYSES OF THE CURRENT DESIGN DIRECTIONS

I. Introduction

Amid the mass of Database design methodologies, a designer must keep in mind that the object of the endeavor is to produce a viable Database design. When so many esoteric topics beckon and call it is easy to be distracted from this destination; to be seduced by the formal perfection of mathematics or fall fascinated into the labyrinthine logic of machine optimization. It is easy to become confused and intimidated in the ubiquitous jungle of theories and jargon. When so much seems complex and confusing one tends to regard with suspicion that which is simple and obvious.

Database design, however, should be, and can be simple and obvious, straightforward and comprehensible. If one does keep the real goal in mind, most of the complexity and the confusion can be dispensed with. This chapter consists of an attempt to cut away and discard that which is superfluous and irrelevant in Database design. It is, if you will, an exercise in the judicious use of Occam's Razor.[1] Design approaches

---

[1]Occam's Razor - "the maxim that assumptions introduced to explain a thing must not be multiplied beyond necessity" - Random House College Dictionary.

will be tested against the touchstone of pragmatism; if some aspect furthers the goal of reaching a viable Database design it will be accepted, if not, no matter how mathematically elegant or machine efficient, it will be rejected.

The analysis in this chapter is roughly patterned after a top-down binary (lion in the desert) search. The approach is to first examine the general design directions. Then, as needed, greater detail is added, until at the conclusion of the chapter, certain specific methodologies are critiqued.

II. A Question of Priority - Man or Machine

Among the current Database design methodologies two distinct and mutually exclusive schools of thought dominate and vie for preeminence: the machine efficiency school and the information oriented (logical design approach) school. Although within each school there exist many factions, before attempting to create order out of the multitude of factions, the fundamental philosophic issue (machine efficiency versus information orientation) must be resolved.

The case for machine efficient design emphasis may seem to be a strong one. Today's Databases push much of the extant machinery to their performance limits.

Many Database systems, for example, work in an on-line environment. Consequently, response time is of paramount importance. Another salient consideration is efficient storage. Typically, Database systems handle large quantities of data. Despite the decreasing cost of storage (both primary and secondary), an inefficient data storage scheme can waste considerable sums of money, (as well as significantly reduce response time). Offhand it would seem that increased software complexity might be a small price to pay for a DBMS which will optimize expensive machine resource utilization.

A problem with this line of reasoning is that there are embedded within it critical assumptions which cannot pass unexamined; implications whose ramifications will come to fruition only at some future date.

One tacit assumption behind machine oriented design is the presupposition that there can exist foreknowledge of such factors as data item access frequency and number of occurrences. Adequate estimates of these parameters might be obtainable for a system at the time the Database is being designed; but what can be said about the unpredictable future? Is not data storage flexibility (adaptability) one of the goals of Database technology? One of the basic insights is that no system is static. What happens to data storage flexibility

when critical factors in the data storage scheme are the access frequency of certain data items and the number of times they are expected to occur? It is almost certain that the addition of a new application to an existing Database so designed would cause havoc; requiring modifications to the extent of a total redesign.

Another assumption of the machine oriented approaches is the continued existence of the machine itself. A Database which is designed around the specifications (idiosyncrasies?) of a particular machine can have no pretense to machine independence. In the best circumstances certain compatibilities will exist between a single vendor's different models; in the worst cases, even the upgrading of an operating system will precipitate serious redesign problems. This is not the place to rehash all the old arguments against machine dependent application software. Many a DP center still bears the scars of lessons learned the hard way in this regard. Unfortunately, except for those who continually live in innocent bliss (faith in the one and only vendor), many of the machine oriented apostles are doomed to repeat this sad and costly history.

Earlier it was stated that "Offhand it would seem that increased software complexity might be a small

price to pay for a DBMS which will optimize expensive machine resource utilization," but this is not so. The most valuable and the most fallible resource at any DP installation is its professional personnel. Software complexity has an exponentially increasing debilitating effect upon professional personnel and their productive output. This is not to deny that extraordinary and talented people exist - people who revel in the complex and the obscure. Highly complex systems have been designed and implemented. Geniuses find their way into the computer field as well as into any other, but systems should not be designed under the assumption that there will always be a genius present to administrate them. To quote Henry Kissinger writing in another context:[2] "The very complexity of these arrangements doomed them. A system which requires a great man. . . sets itself an almost insurmountable challenge. . . " (1971, p. 921).

Machine oriented designs are, by their very nature, logically more complex than designs formulated in accordance with human modes of thought. The end product of an information system is information. A Database is a tool within an information system, a

---

[2]Describing Bismarck's web of international alliances.

means to an end (and not an end in itself). The whole system, and all its tools should be as conceptually simple and comprehensible as possible.

It has been said that: "Man is born free, and everywhere he is in chains." Men created machines to serve the needs of men. How absurd it is to find some men now serving (orienting themselves towards) the needs of the machines. It is, to say the least, psychologically demeaning.

Given the state of the art of today's Database technology, even the most fanatic supporters of the information oriented approach would concede that when it comes to the actual, practical, final design and implementation of a Database, response time and machine utilization must be taken into account. The machines are simply not "sophisticated" enough to allow the designer to ignore such considerations. The real question, however, is whether in order to achieve an "optimal" Database design, one must begin the design process by defining the physical machine oriented characteristics. This author's answer is emphatically "no". A machine efficient Database design approach is not a guarantee that the final product will make optimum use of even the machine's resources. In many cases a machine oriented approach results in serious sub-optimization

problems (i.e. the optimization of one specific resource to the detriment of the utilization of the total resources). Certainly in regard to flexibility, adaptability and the "expensive" human factor, there are considerable problems with the machine oriented approach.

The logical design based approach falls prey to none of these shortcomings. It places the machine in rightful subordination to the considerations of logical information content. The information oriented school itself, however, is not a unified whole. Some of the factions, and their arguments will be examined in the following sections.

III. Logical Schema Design - The Specific Versus the General

When the analyst inaugurates the Database design at the logical design (information oriented) level a problem similar to the machine-oriented versus information-oriented distinction arises. It is the problem of specific package orientation versus generalized package-independent design. Stated simply the question is whether to begin designing a schema to fit the mold of a specific DBMS package, or to initially construct a package-independent data (information) model and then from it derive a specific schema (by

imposing on the general, unrestricted model, the idiosyncratic restrictions of the package).

In actual practice the specific package oriented design is quite prevalent. It is natural when using a specific package to begin thinking in the terms of that package and in line with its mode of structure. An analyst working with hierarchical data structures begins to think in terms of data hierarchies. He soon begins to intuitively organize data interrelationships into hierarchies and they become for him the natural way to think. In like manner, the analyst who works with networks or with relations begins to feel comfortable and natural with the type of data structure to which he has become accustomed. This is, in many cases, the real basis for what was discussed in Chapter Two as being the "Artistic Intuitive Approach".

A package-oriented design by a designer familiar with the package can be a very efficient operation. If the specific DBMS package is powerful and flexible, the resulting Database will probably be powerful and flexible. On the other hand, a restrictive DBMS package will almost certainly cause the designer to create a restrictive and convoluted Database. Regrettably, even though some Database packages may appear to give the designer a significant degree of conceptual freedom, all existing (and

foreseeable) DBMS packages are to some extent conceptually restrictive. This is a serious drawback to the package oriented design approach.

A second drawback which exists today (but which may be diminished in the not too distant future) is package dependency. Unfortunately, being package oriented usually means being package dependent. Like machine dependency the effects of package dependency are generally seen only over an extended time period. How foolhardy it is to irrevocably tie oneself (and one's company) to a vendor if it can at all be avoided. No matter how powerful the vendor may seem, the sands of time have toppled many a mighty giant. Far better to face the vicissitudes of fortune unencumbered and flexible than needlessly tied to the fate of another.

The hope for the future in this situation is the growing interest and acceptance of standard DBMS features. More and more vendors are turning towards the CODASYL DBTG recommendations, and ANSI seems close to endorsing them. This trend should lessen the danger of package dependency. Eventually standard packages may be as compatible and transportable as ANSI standard COBOL programs. For the present, however, it must be remembered that widespread standard DBMS packages are still a feature of the future and not a virtue of our own time.

Thus, package oriented design, widespread though it may be, contains two serious (fatal?) flaws: restricted conceptualization and package dependency. The alternative, general package independent design has neither of these faults. By beginning the design process at the conceptual level, the designer is free to formulate his ideas in direct response to the information dictated data relationships, the "inherent" data organization. Free from the pragmatic confines of specific DBMS packages such a model can be a truer reflection "of reality" than any restricted model. Free from the idiosyncrasies of a specific package the unrestricted conceptual model is eminently transportable. Simple design compromises can adapt it to any reasonable DBMS package and when the time comes to transport the design, the Database, to another package, the original (uncompromised) design is still valid.

It is hard not to find the idea of a conceptual level design appealing. Yet if it is so appealing, why isn't it more widely used? One reason is that the concept of a "conceptual schema" is still relatively new. Another reason is that although it is easy to say that one should work from a "conceptual model" there is very little agreement as to how to formulate that model and what modeling tools should be used. This lack of

unanimity is evident even from the brief survey of techniques described in Chapter Two. In the following section an attempt will be made to critique the current conceptual level approaches, and, barring a full endorsement of any one particular approach, at least an understanding of the current shortcomings will be reached.

IV. Logical Schema Design - Problems with the Functional Approach

A conceptual schema is often called an enterprise schema,[3] that is, it is supposed to be a model of the enterprise it serves. There are, however, a number of ways in which the reality and the information needs of an enterprise can be modeled. One method is to take the total information picture as a gestalt, an integrated whole of interrelated data. An alternative method is the Functional Approach, a model based upon the enterprise's functional features. Although the functional approach is a direct descendant of the very powerful and successful top-down design technique, there are some serious problems with it.

A Database is, by its very nature, an information integrator: a means whereby the information needs of all

---

[3]Not to be confused with the E-R approach's specific use of the terms "enterprise schema".

its users can be stored in an integrated manner. Paradoxically, in order to store information in an integrated flexible manner, one stores data and not information per se. There are really two ways in which information can be stored: implicitly and explicitly. To store a unit of information explicitly is to group all its component data into one "lump" (or record). To store information implicitly is to store the component data independent of a specific information context, but in such a manner that the desired information can be constructed out of its components when need be. Considerations of maximum flexibility demand that as far as physically possible, information should be stored implicitly. Only those datum which are inherently connected (such as a person and his social security number) should be stored together (in one record).

What the functional approach does, by its process of dividing the information into fundamental units of information, is create a collection of records (groups of data types) each containing a basic unit of information. This scheme is very sensible in the traditional file environment, where one is interested in storing information explicitly. It is not, however, very suitable to Database design. There is, for example, no guarantee that different functions may not need the same piece of data, but in different information contexts.

By following the functional approach such a piece of data would appear in multiple records, a problem of data redundancy. Similarly, the practice of storing explicit information requires a serious redesign effort whenever a new type of information is needed (even if all the requisite data are already in the "Database"!).

This is not to entirely repudiate the value of the top-down techniques as an aid in Database design. Determination of the information of an enterprise is basic to Database design. Top-down, functional analysis provides a powerful method of ascertaining an enterprise's information needs (and thus the information needs of its Database), but this should be the extent of the functional analysis role. It is a useful tool, but out of place when used as a schema designing implement.

V. Conceptual Level Design - A Search for Simplicity

In the introduction to this chapter it was stated that "the object of the endeavor is to produce a viable Database design". Some readers may find the phrase "viable Database design" to be somewhat vague and the time has come to expand upon that idea and make it more specific. Despite whatever fascinations Database concepts may hold for the purely academic world (and other interested bystanders) the subject finds

its origins deeply embedded in the practical, the applied world. More than anything else a Database is a tool. Its value lies in its ability to be used. Aesthetic considerations, no matter how intellectually soul statisfying, can only be of secondary (almost insignificant) importance.

Like any tool its usefulness is seen in its power to perform a job well and its applicability, or adaptability to handle a wide range of situations. A designer of tools must always bear in mind, among other things, who it is that will use the tool. A thirty kilogram hand-held hammer for example, might certainly be able to pound nails with considerable force but its usefulness would be extremely limited, both by reason of its lack of applicability to many situations (delivering too much power) and by reason of its unwieldable weight.

A methodology for Database design is also a tool; a tool used to design another tool. Just as a Database is primarily a practical tool, by association, the Database design tool must also be primarily a practical tool. As such, the authors of Database design methodologies must take into consideration the identity of the users of the tool; the design artisans.

The Database design artisans are for the most part DP professionals with considerable experience in

the field. They are system analysts and programmers who have risen to a position of precedence. They are not primarily mathematicians, logicians, semanticists and philosophers. Why then are so many Database design methodologies geared to mathematicians, logicians, semanticists and philosophers? What good does it do to create a Database design methodology which is comprehensible only to a specialist (or at least a cognoscenti) in one or more of these peripheral disciplines?

Many of the "conceptual level" design methodologies are of this ilk. Consider the "infological approach" and "semantic predication analysis". The infological approach, with its mixture of symbolic logic, semantics, relational mathematics and realist philosophy, is perhaps an extreme case of erudite design. Semantic predication analysis, in addition to being equally obscure, is also of questionable value. Typically "word pictures" of an organization and its functions are far less precise (far less informative) than diagrams and schematics. If a complete word picture of an organization can be composed, then most certainly a schematic can also be composed with far less effort.

Even the relational approach, with its emphasis on the not so mundane relational mathematics and its "confusing language" (Martin, 1976, p. 96) cannot hope to be comprehensible to the average system analyst. The

proliferation of "normal forms" (so reminiscent of Ptolomy's proliferating epicycles) can only cast a doubtful light upon the relational model's claims of logical simplicity and ease of use.

There exists just one justification for such esoteric Database design techniques (aside from the desire to create a white elephant and/or the need to churn out pulp for the publisher's mill): that there can exist no simpler, less abstruse, method of accomplishing the same task. But, if there does exist a simple Database design methodology, one which with far less complexity reaches the same goal, then it follows that (with a swipe of Occam's razor?) all the abstruse methodologies should be summarily discarded.

The issue, thus, centers upon the question of whether a simple and comprehensible Database design methodology does exist. Without the existence of such a methodology Occam's razor becomes duller than a butter knife. Both the canonical Database design and the Entity Relationship approach are simpler than those mentioned above (comprehensible to the average DP professional) but, as will be seen in the next section, they are not ideal alternatives. The E-R approach is still overly complex and the Canonical design is not purely a conceptual level design. Nevertheless, Chapter Four of this

thesis emphatically asserts that a simple, easy to use, yet powerful Database design methodology does indeed exist. Before expanding upon that topic, however, the assertions made above concerning the canonical and Entity Relationship approaches must be justified.

VI. Two Serious Design Methodologies - A Closer Look

Of all the Database design methodologies which this author has encountered, the canonical design technique and the E-R approach have left the most favorable impressions. Each is an information oriented approach. Each is not overly erudite. In the absence of a better tool, each would probably do a passable job. The problem is that each one of these methodologies is, in its own way, somewhat deficient.

A. The E-R Approach

The E-R approach is a true conceptual level approach. It is not concerned with serving the particular structure of any given DBMS package, but rather its goal is to be an accurate reflection of "reality", one which can then be used as the basis for a package oriented design. If it were to be judged only upon whether it reaches its goal or not then the E-R approach must be deemed a success. Unfortunately, how one reaches a goal is often as important as the goal attainment

itself. The E-R approach does provide a means of modeling "reality", but both the construction of the model and the finished model itself are far more complex than they need to be.

The E-R model makes use of entity sets, relationships, attributes, value sets, and upper and lower conceptual domains. How well defined are these concepts? Are they all necessary? Consider the problem of the enterprise administrator, the person responsible for determining the "entity types most suitable for his company." How does he identify an "entity"? Peter Pin-Shan Chen provides the following guide: "An entity is a 'thing' which can be distinctly identified." (Chen, 1977a, p. 17)

Intuitively everyone knows what a "thing" is. A "thing" exists (if not physically then conceptually) and can be described. It is the opposite of "nothing" (which cannot be described). In other words, a "thing" is that which has attributes. Thus, upon first glance at the definition, one is led to believe that an "entity" is that which has (or can have) attributes (or properties).

This expanded definition of an entity must, however, be short lived. Further examination of the E-R approach discloses the following: "Entities and relationships have properties, which can be expressed in terms of attribute-value pairs" (p. 17). If both entities and

relationships can have properties (attributes), then both must be describable, and hence both must be things. How is one to distinguish between entity things and relationship things?

The answer to this question is implied by the following excerpt from the definition of relationships: "Relationships may exist between entities" (Chen, 1977a, p. 17). From this it is deducible that entities must be unitary, indivisible things while relationships are combined, or concatenated things. This confusing situation can perhaps be clarified through the use of an example.

Suppose that the local electric company keeps customer records keyed on a customer service number. A customer service number is a unique identifier of a customer. As far as the system is concerned, a customer is a unitary indivisible thing, thus the customer service number can be thought of as identifying an entity. As the unique identifier of the entity, the customer service number itself can be thought of as being the entity (which in turn may have attributes, e.g. name, address, phone number, and so on). Other entities which the system might include are billing cycle number, branch office number, service route number, etc. Each is, in some sense a unitary thing and hence each is an entity.

It seems now that the problem of identifying entities has been clarified (although as shall be seen shortly, it is not as clear as one might wish). The next natural question is: How does one identify relationships? If a relationship is a concatenated "thing", it should be easy to identify. Consider again the example of the electric company. It retains, as has been noted, customer service numbers. Each customer service number is made up of, among other things: the branch number, the billing cycle number, the service route number, and an address code. The multiple concatenation of all these parts results in a unique customer "service number". Thus the customer service number is a relationship. As a relationship it is entitled to have attributes (such as customer name, address and phone number).

It seems therefore that the original analysis, which declared a customer service number to be an entity, was in error. Obviously, it follows from the definitions that a customer service number is a relationship. All the steps in this argument have led logically from one to the next and the conclusion is inescapable. Nevertheless, this author submits that the conclusion is, from an intuitively logical point of view, an absurdity. The customer service number uniquely identifies a customer. Is one to assume from this that one

must think of a customer as a "relationship"?

There is an alternative to this line of thinking, one which more closely follows the normal pattern of human thought. Would it not be far simpler to postulate only one generic term to describe "things"? The human mind deals with concatenated "things" as "entities" all the time. A person's full name, for example, is really a concatenation of his first and last name (and any others which might be appended). The E-R approach of splitting "things" into two ill-defined and unnatural groups can only lead to confusion.

The example above is not the only instance of over complexity to be found in the E-R approach. Consider the postulated capacity of values to become entities and entities to become values. Entities and values each inhabit a different conceptual domain (entities the upper, values the lower). Recognizing that occasionally something which has always been thought of as a value needs to be thought of as an entity (and vice versa) the E-R approach provides a process for transferring a "thing" from one conceptual domain to another (it is one of the five basic modifying operations). For example, suppose office number has always been thought of as the value for the attribute office of the entity employee. One day the need arises to take inventory of the offices.

Thus it might be useful to think of employee as the value of the attribute resident of the entity office number.

It is clear here that "things" can be either entities or values with equal facility. In other terms one might say that a thing is sometimes an object (a possessor of attributes) and at other times it can be thought of as an attribute of some other object. No one "thing" is uniquely an object, just as no one "thing" is uniquely an attribute. Why then, must a division into conceptual domains be made? Why should the rigid distinction exist between "entities" and "values"? They are, after all, two sides of the same coin. It is far simpler to postulate the existence of only one generic term to cover all the E-R concepts. Such a singular concept could be seen as being usable in different modes: as an object with attributes, as an attribute of an object, as a component of a concatenation, and as being either indivisible or concatenated in its own right.[4]

The point is this, that the E-R approach has postulated far more concepts and complexities than are

---

[4]Just such an approach will be introduced in Chapter Four.

really necessary to deal with the problems at hand. Even the relatively simple E-R diagrams become hopelessly tangled and interwoven in a system of even moderate scale. It is almost as if the E-R approach suffers from ivory tower naivete; it is an intellectually stimulating idea, but of questionable practical worth.

B. Canonical Database Design

The Canonical Design technique, like the E-R approach, is a methodology for a package independent design. The object of the design is not a "working" schema (one which is immediately usable with a particular DBMS package) but rather a generalized schema design which may be modified to meet the specific requirements of a specific package. The canonical design methodology purports to be able to discover the "inherent properties in the data which would lead to data items being grouped and groups being interconnected in a particular structure" (Martin, 1977, pp. 248-49). This is an admirable goal; unfortunately the canonical design approach does not reach it.

Although attempting to divorce itself from the practical restrictions of today's DBMS packages, the canonical design is too steeped in current DBMS package

traditions to be considered a truly "conceptual" design approach. Definitely not a purely package oriented design, the canonical design inhabits the quasi conceptual, quasi package oriented middle ground between the two approaches. What causes this slight package orientation seems to be a genuine concern for practical applicability. In many respects the canonical design's practical approach is a definite asset over some of the ivory tower conceptual approaches. The use of the convenient bubble charts as a schematic and explanatory base indicates a genuine concern for widespread comprehensibility. The automated package (IBM's Data Base Design Aid - DBDA - which incorporates features of the canonical design) renders the canonical approach feasible even in highly complex systems (circumstances which would make incomprehensible spiderwebs of the most carefully drawn bubble charts). Nevertheless, the restrictiveness and the narrow horizons which ultimately plague all package oriented designs, prove also to be the canonical design's undoing.

One minor restriction which characterizes the canonical design is its prohibition of M:M relationships. It is true that most of today's DBMS packages will not support M:M relationships, but this does not mean that

an M:M relationship[5] is not occasionally an important conceptual construct. Sometimes an M:M relationship is a true and accurate picture of a situation (as, for example, in the case of authors and their books: one author may have many books, while one book may have many authors). If one is trying to model the "inherent" structure of information, then one must allow for M:M relationships.

A second, more crucial problem, with the canonical design is its use of "secondary keys" (or secondary indicies). In forming the bubble chart, the canonical design recognizes "candidate keys" i.e. data items which are used to identify other data items. Data items which are non-key items are termed attributes. In another terminology the candidate keys would be called objects which in turn possess attributes. This is a classical conceptual distinction; one which is intuitively valid, even to non-DP initiates. Canonical design, however, then proceeds to introduce into the picture a third category of data item, the secondary key. In the process of converting the bubble chart into a "conceptual schema" diagram, the class of candidate

---

[5]In the context of the canonical design. In the Chapter Four methodology the M:M relationship is seen in a different context.

keys are split into primary keys and secondary keys. A primary key is an object which possesses single valued attributes. A secondary key is an object which possesses only multivalued attributes. From a conceptual point of view such a distinction is totally incomprehensible.

If, for example, it is postulated that cars can be only monochromatic (i.e. the object car can have only one color attribute) then cars would be primary keys. If, however, cars could be considered polychromatic, then cars would be relegated to the status of secondary keys. Where is the logic behind such a state of affairs?

To a DP professional the distinction is not without reason. Often it is more convenient to retain multiple valued attributes on a separate (secondary) index file. Hence, if an object has only multivalued attributes it becomes pragmatically reasonable that the attributes might be better handled through the use of a secondary index. What, however, have such considerations to do with a conceptual design? They are out of place and inappropriate.

In, and of itself, this lack of adherence to "pure" conceptual design is not fatal. If the introduction of a few practical considerations is required to enhance the viability of a design, then they should not

be rejected merely upon the grounds of "conceptual impurity" and complexity. Regrettably this is not the case. The introduction of the canonical design's package oriented features renders its product less powerful, and less viable. The schema which the canonical design generates must often be considered sub-optimal for a number of reasons. Although this is not the place to delve into the problem in greater depth, the interested reader is referred to Appendix D, wherein will be found an analysis of an example given by Raver and Hubbard in their exposition of the canonical design approach. Included in that analysis is an alternate schema design generated by the method to be expounded in Chapter Four of this thesis.

Thus it is that the canonical design, having set for itself the worthy goal of all conceptual design approaches (the generation of an "information" model), it falls short of that goal. A package-oriented tint to the canonical technique has impeded its progress in what this author believes to have been a start in the right direction. Building upon many of the basic ideas of the canonical design it should be possible to reach the goal, to create a pure information model. Chapter Four of this thesis will explain just how it can be done.

CHAPTER FOUR

A PROPOSED NEW DATABASE DESIGN METHODOLOGY

I. Introduction

The subject of a Database is information. It is the end for which the Database exists. A computer can be regarded as a "black box" which enables men to store data and (if properly used) to retrieve information. Both data and information have an existence of their own; quite apart from any specific computer or software package.

The function of a Database is twofold. (1) It must receive data from the external world and instruct a specific computer how that data is to be stored. (2) It must produce information upon demand by instructing the computer to retrieve the appropriate stored data.

To fulfill its dual role the Database must interact in two distinct directions. (1) It must be able to interact with a specific machine (the hardware and the supporting software) in order to govern the storage and retrieval of data. (2) It must contain a map of information (the data and the interrelationships between the data which comprise information).

The Database-machine relationship is machine-dependent and transient. It has no existence apart from a specific machine and its specific environment. The

Database's information map is machine independent and eternal. As long as the information it maps is valid, even if no machine exists, the information map itself (the data relationships which form the information) is also valid.

In accord with the dual nature of Databases, Database design should consist of two discrete steps: the design of the information map (alternately known as a Database logical design or conceptual schema), and the design of the Database-machine interface. Insofar as one of the purposes of a Database is to provide flexibility (which should include machine independence), and insofar as one of its paramount characteristics should be logical simplicity ("easy to use"), it is obvious that the design process should begin with the information map. Once the information map is designed it can be modified to meet the restricting requirements of any specific machine and DBMS package. In this manner, machine dictated modifications to the innate logical design can be easily recognized as such; and not confused with the essential underlying information structure.

The methodology described in this chapter is a technique for constructing information maps. It begins with a discussion of the theory behind the method and

concludes with an explanation of the methodology itself (including some considerations on how to handle machine imposed restrictions).

II. The Theory Behind The Method

Specific information consists of specific instances of data and specific relationships between data. The piece of information "the house is blue" conveys a message about a specific object (the house) and a certain attribute of that object (its blueness). An information map, however, does not deal in specifics, but rather in abstractions. It is irrelevant to the information map whether any given house is blue or green. What does concern the information map is that a house, a kind of object, can have a certain attribute called color. The information map deals with objects (i.e. possessors of attributes), attributes, and relationships (the connection between objects and attributes). If one were interested in information about a house, its color might be a requisite aspect of that information.

At this point the temptation is strong to immediately classify the world into "objects", "attributes" and "relationships". This division, however, is an ephemeral one. What at one instant is an object can in the next be seen as an attribute. What at one instant is an attribute can in the next be seen as an object.

Consider the statements "Dan's house is blue" and "blue is sad". In the second statement, that which was previously seen as an attribute of house, "color", (blue) is now an object possessing its own attribute, "mood", (sad). In the first statement, that which was only an object "house" is now both an object (for it still possesses the attribute color) and an attribute (for it is possessed by - is an attribute of) the object "person".

Thus it is that by nature information contains only <u>two</u> basic components:

1. generic concepts which can be abstracted from specific instances of data (e.g. color, employees#, mood, height, salary, project#, etc.)
2. relationships between generic concepts.

A generic concept is called an <u>eidos</u>[1] (the plural is <u>eide</u>). The eide must play both roles: "object" and "attribute"). Any eidos can be considered either an object or an attribute depending upon the information context in which it is used.

One must be careful in working with eide. It is easy to assume that for each eidos a recoginized word

---

[1]An eidos is an abstracted generic concept (literally "form). According to Plato, men recognize things for what they are due to a physical object's resemblance to its eidos (e.g. a physical chair is recognized as a chair because it partakes of "chairness", that is, in some way it resembles the eidos of chair).

exists. This is definitely not the case. An eidos can be any concept which is used either as an object or as an attribute. An example of an eidos for which no single word exists can be seen in the following case.

Suppose a company purchases a certain item (let it be called "XYZ") from a number of different suppliers, each of whom supplies many different types of items to the company, and each of whom charges a different price for xyz. Within the company, items are identified by a unique PART#. In attempting to obtain the piece of information, "what does supplier 'q' charge for xyz?", one discovers that neither the PART# of xyz, nor the name of a supplier is sufficient to identify the information being sought. The price cannot be considered a function (an attribute) of PART# because different suppliers use different prices. Similarly price cannot be considered a function of SUPPLIER since each supplier supplies numerous items. It turns out that in this situation price is a function of the concatenation (the linking or logical intersection) of SUPPLIER and PART#. No single word exists for this concatenation. Nevertheless, the eidos here must be the concatenation "PART# - SUPPLIER", for <u>it</u> is the <u>true</u> object of the attribute "price".

Not all concatenations require new, unusual, or hyphenated names. In regard to persons, "name" itself

is often a concatenation (although it is rarely thought of as such). What is often meant by "NAME" is a person's full name, a concatenation of at least his first and last name (and possibly a middle name as well). It is important to realize that concatenations are common eide, having no greater nor no less a claim to serve as either object, or attribute, than the logically indivisible eide.

The relationships between eide which concern a Database information map are solely "relationships of attribute". This means that either a certain eidos can have another as its attribute or it cannot. If it can, then the relationships can either be one of unity (1:1) or one of multiplicity (1:M).

Although it is common practice to speak of M:M relationships, an M:M relationship is a misnomer. In strict terms, a relationship exists between an object and its attribute (i.e. between one object and one or more attributes). The object side of the relationship must always be a "1". What is usually seen as an M:M relationship is in reality <u>two</u> complementary relationships. Consider the example of the two eide book and author: <u>one</u> book may have many authors and <u>one</u> author may have written many books. In this classic M:M situation it can be seen that the M:M relationship is

really two relationships: one from book to author and one from author to book. Each relationship is a 1:M relationship, but no M:M relationship exists.

A relationship can either be named or unnamed. In some instances a named relationship takes on the name of the attribute (e.g. the relationships between the eidos "PERSON" and the eidos "AGE" would most probably also be called "AGE"). In other instances a relationship may possess a descriptive name of its own (e.g. a relationship between the eidos "MAN" and the eidos "WOMAN" might be termed "SPOUSE").

It should also be noted that more than one relationship may exist between a single pair of eide (e.g. if an airplane is to use an airport as an intermediate stopover, there will be two relationships between the eidos "AIRPORT-FLIGHT" and the eidos "TIME": "ARRIVAL-TIME" and "DEPARTURE-TIME").

It is obvious that the realm of the eide and their interrelationships is limitless, as indeed are the possibilities for information. It must be understood that for practical reasons any given Database's information map can contain only a small subset of the set of universal information. Although any specific eidos can be thought of as both object and attribute, it will happen that within the subset of a given Database some

eide will be used only as objects, some only as attributes, and some as both.

There are many ways to represent an information map. As with most pieces of complex software, the representation which is "machine readable" (even a high level language structure) is usually not the most comprehensible to humans. In the case of an information map (an entity which should be machine independent anyway) the most convenient format is probably a diagram.

The diagramatic conventions which are used in this methodology are not original. Similar conventions are common. To avoid ambiguity, however, definitions in the light of the above "theoretical" discussion, are provided. The following are the conventions used with the graphic reproduction of schemata:

a. All eide are represented by a rectangular box which contains the name of its eidos inside.

Figure 4.1 Eide

b. Relationships between an object and those eide which are used soley as <u>its</u> attributes can be of two types, 1:1 and <u>1:M</u>. In either case the attributes are aligned in a contiguous row to the right of the object. In the stance of a 1:m relationship two extra partial boxes are drawn "behind" the

original attribute's box (as shown in the illustration below with the eide ALIASES and IDENTIFYING-MARKS).

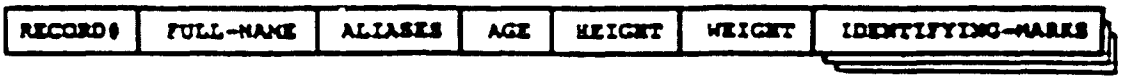


Figure 4.2 An Object and Its Unique Attributes

c. Relationships between an object and those eide which are used either as objects themselves (in other relationships) or as shared attributes (i.e. as attributes of more than one object) are represented as arrows. The direction of the arrow always indicates the direction of the relationship, running from the object to the attribute. 1:1 relationships are single headed arrows, 1:M relationships are double headed arrows.

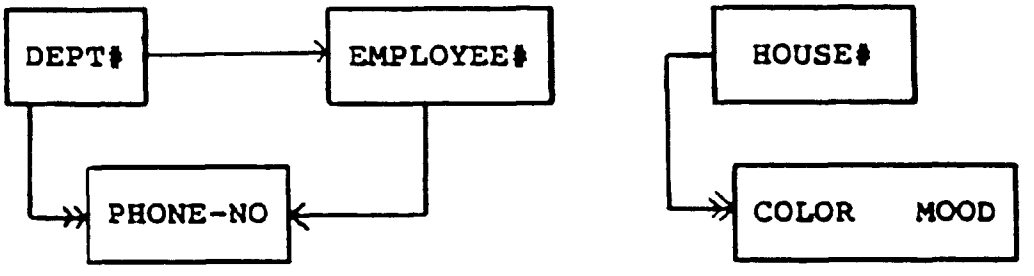


Figure 4.3 1:1 and 1:M Relationships

d. When two eide are related by two relationships such that the object of one relationship is the attribute of the other relationship, instead of using two arrows (which might tend to clutter a diagram) both relationships are represented by one-bi-directional arrow. It must be remembered that relationships are always "one way" (from object to attribute) arrows. Thus all bi-directional arrows indicate the presence of two complementary relationships. Figure 4.4 illustrates this convention; while Figure 4.5 presents some examples.

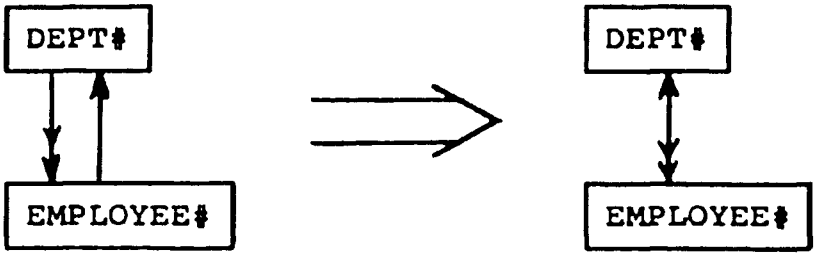


Figure 4.4 Notation for Complementary Relationships

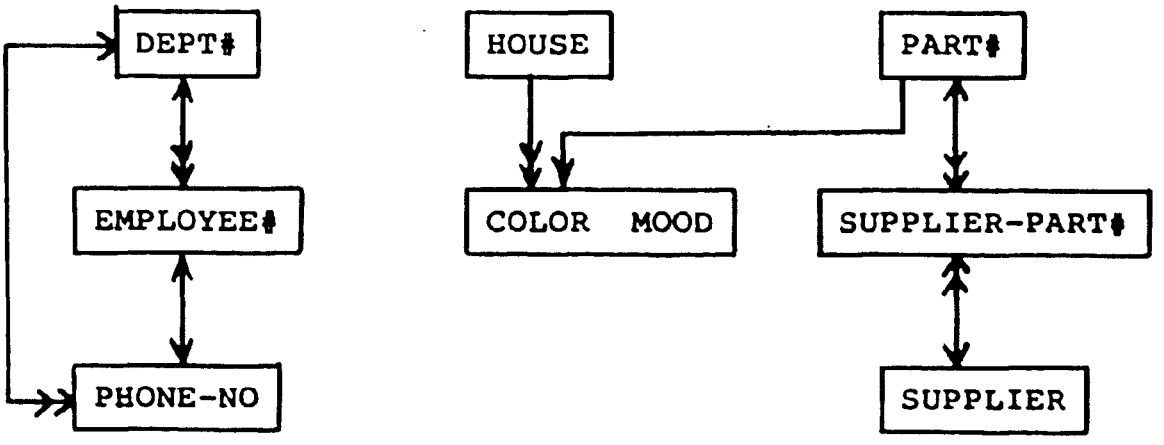


Figure 4.5 Examples of Notational Conventions

e. In the event that a relationship is a "named relationship" the relationship name is written either intersecting the arrow or along side it. When two relationships share an arrow (and it is not clear to which relationship the name belongs) a small arrow can appear next to the name.

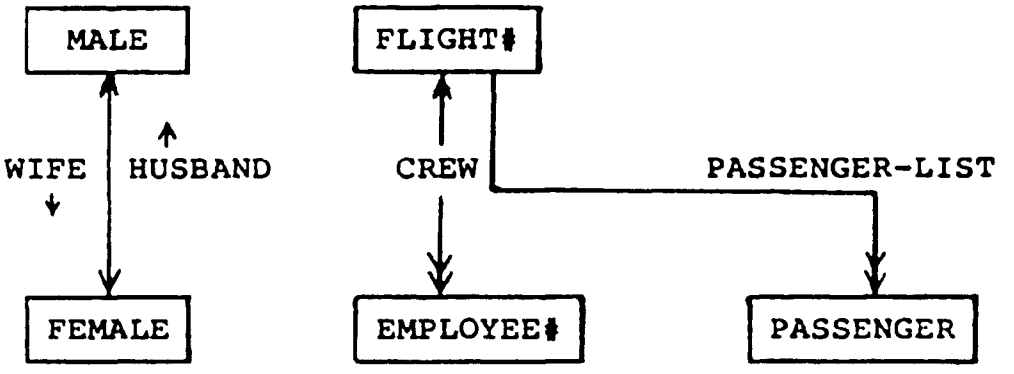


Figure 4.6 Named Relationships

An information map is an abstract model of information. It is not a "working"-schema (i.e. one that can be directly applied to a specific DBMS package). Given the current state of the Database art, there should be a distinction made between conceptual schemata (unrestricted conceptual models of information) and working-schemata (models restricted by specific package limitations).

In order to maximize machine and package independence, Database design should always begin at the conceptual schema level (not at the working-schema level).

Working-schemata are derivable from conceptual schemata, but the reverse is not always true. Using a valid conceptual schema as a base (a pure information model) a designer can knowledgeably modify the model to accommodate the pragmatic restrictions imposed upon him. Parameters such as response time, access frequency, controlled redundancy (how much and at what cost), access modes and so forth can all be taken into account and used to "tune" the working-schema. A properly tuned working-schema is essential to a successful Database design. It is the short term end product of the designer. It is the interface between a machine and the conceptual schema, the legitimate child of the conceptual schema's union with practicality.

The most convenient vehicle for a conceptual schema model is the information map (the "object-attribute-relationship" diagrams). This is an effective model, simple yet powerful. The format comes easily to DP professionals (who can, for example, think of eide as data items). Unlike the relational approach (and some others which only a mathematician could love) the information map deals in a terminology which is comprehensible. It is natural for both DP professionals and non-professionals to think in terms of objects which have attributes and which may be related. All of which

(objects, attributes and relationships) are clearly discernible within the model. As will be shown in a later section of this chapter, the information map is an easy model to construct (a significant portion of its design procedure can be automated) and that a conceptual schema is easily convertible to a working-schema format (resembling as it does the CODASYL DBTG schema diagrams, conversion from an information map to a DBTG type schema is almost trivial).

Before turning attention to the practical questions of how to actually design a conceptual schema and how to convert a conceptual schema to a specific working one, here is a brief review of the basic tenets of the theory "behind the method".

1. The primary concern of a Database is information.

2. The Database must act as an interface between man and machine: accepting data from man, storing data in the machine, and retrieving information for man.

3. The primary concern of the Database designer should be the Database's information map (its eternal soul) and not a specific machine environment (its mortal body).

4. The basic building blocks of information (and thus the features to be found in information maps) are eide and relationships. Eide have two functions: to be objects and to be attributes. A relationship connects between object and attribute; it is that, by virtue of which, an object is an object and an attribute is an attribute.

5. Database design should consist of two major steps:

   a) design of a conceptual schema (the information map)
   b) modification of the conceptual schema to include machine necessitated restrictions.

6. The most suitable model for a conceptual schema is an information map (an object-attribute relationship model).

III. The Design of a Conceptual Schema

The design of a conceptual schema consists of two phases. These are:

Phase I - identification of the information to be modeled

Phase II- construction of the information model (the conceptual schema) from the identified information.

Each phase is an integral part of the process, and each is of equal importance. Phase I is the more intuitive phase. It requires experience, as well as an intimate knowledge of the systems involved in order to accurately identify the information needs and data interrelationships of the total Database environment. Phase II is the more mechanical phase and is the primary topic of this thesis.

In the following discussion, Phase I is described briefly; more from the point of view of what results it should yield rather than the manner in which those results can be obtained. There are, after all,

many valid ways of identifying the information to be modeled. Phase II, on the other hand, is the crux of the methodology being presented and is thus dealt with in significant detail.

A. Phase I

Identification of the information to be modeled consists of identifying the eide (the generic concepts, or data item types) and identifying the interrelationships between the eide. There are many methods to accomplish this task and many tools to aid the analyst in the endeavor. The choice of approach and the choice of tools depend in large measure upon a specific system's environment. An analyst working on a new and nebulous system would naturally proceed differently than an analyst working in a well-defined environment. Likewise the analyst with access to a powerful data dictionary can approach the problem with less trepidation than his less fortunate counterparts.

Regardless of the environment however, there are certain basic elements which must be common to all. One basic element is the ability of the analyst to abstract the relevant aspects of a given system. The success of the operation as a whole depends upon the accuracy of the analyst's perception. The "aspects" of the system which the analyst must perceive are listed below. These

are the minimal information requirements for a successful Phase I.

Phase I Information Requirements

1) Identify and name all eide (generic concepts) which the system is to use. This entails the identification of all the types of data to be found anywhere within the system.

2) Remove synonyms and homonyms from the system. This is a very crucial and delicate step. It requires a full understanding of the concepts to be used. Where an eidos has been given multiple names, (synonyms) one unique and universal name must be decided upon. Where multiple eide have been given the same name, (homonyms) each eidos must be assigned a thoroughly unique name.

3) Identify the origins of the eide. Certain eide are basic to a system and are, of necessity, inputed into the system. Other eide are derivable quantities and need not be system inputs. Such derivable eide are usually the result of a calculation (e.g. the total number "on order" of a given part can be obtained by summing the number requested in each of the unfilled, "outstanding", orders). It is up to the analyst to decide which derivable eide to include in

the schema and which to exclude. Each inclusion brings with it both implicit data redundancy and additional complexity (the fewer the number of eide, the simpler the schema). In general, unless a certain derivable eidos is of paramount importance, derivable eide should be excluded from the conceptual schema. This does not, however, mean that they will be excluded in the final practical design. There are serious practical trade-offs to be considered. These will be discussed in section IV (machine considerations). The point here is that an accurate identification of the nature of a particular eidos must also include information on the possible and most reasonable origins of that eidos.

4) Identify the interrelationships between eide including the "type" of relationship (i.e. 1:1, 1:M). Typically this information will be gleaned from various "users' views" of the system data. A user's view can come from the file organization of an existing system, or from a detailed description of the output requirements for either a new or an

existing system. In performing this identification process great care must be taken to establish the true object and the true attribute in each relationship. Often an existing file structure can be deceptive in this regard. Consider the example of the following record from a parts inventory file:

Figure 4.7 An Inventory Record Description

Even though the file may be defined in the system as a parts information file, PART# is not the only object (key) being described. SUPPLIER and DESCRIPTION are unequivocally attributes of PART#. SUPPLIER-ADDRESS, on the other hand, is definitely an attribute of SUPPLIER, while UNIT-PRICE can be either an attribute of PART# or an attribute of the concatenation of PART# and SUPPLIER. If a given part can only have one price, regardless of who the supplier is, (an unlikely circumstance) then UNIT-PRICE may be thought of as only an attribute of PART#, otherwise it belongs to the concatenation.

Similarly, problems exist in determining the

types of relationships expressed. In the record format above the relationship types are ambiguous. Does, for example, the system allow multiple suppliers for a single part? Does it allow for multiple parts to be supplied by a single supplier? Is the "supplier-to-part" relationship pertinent to the system? The answers to these types of questions are not always easy to obtain; such answers usually require an intimate knowledge of the systems involved. Nevertheless such knowledge is a necessity if the Database logical design is to be a true model of the systems' information requirements.

These are the tasks which Phase I must perform. Depending upon the environment, there are many ways in which an analyst can accomplish them. It is not the intent of this thesis to delve in depth into the problems and methodologies of Phase I. The tasks and some of the problems are mentioned simply as an introduction to the prerequisites of Phase II.

The success of Phase II is not directly dependent upon the format of the Phase I results. It is irrelevant whether the analyst chooses to use bubble charts, 3NF relations, a machine based data dictionary, or any other convenient data description tool. The point is, however,

that in order to engage in a meaningful Phase II, accurate Phase I results must be available.

Before turning to a discussion of Phase II a brief digression concerning notation is in order. As stated above, the format of the Phase I results is irrelevant to the success of Phase II, however, the analyst must be careful to avoid an ambiguous format. The classic rectangular record format is just such an ambiguous format. For use with the examples in the discussion of Phase II, this author has chosen the bubble chart format. Although possibly not as familiar to most analysts as the rectangular record formats, bubble charts are by far more expressive.[2] Consider Figure 4.8. It has none of the ambiguity of Figure 4.7's record format. Undoubtedly, the analyst must possess a more detailed knowledge of the system in order to draw a bubble chart, but, as mentioned in the previous paragraph, this is knowledge which must, in any event, be obtained.

It should be noted that in the bubble chart Figure 4.8, the relationships[3] between PART# and SUPPLIER

---

[2]The bubble charts used here represent each eidos (regardless of its context) as an elipse and represent relationships with the same "arrow" conventions as are used in the schemata (see Chapter Four, section II).

[3]Not an M:M relationship, but two distinct and complimentary 1:M relationships.

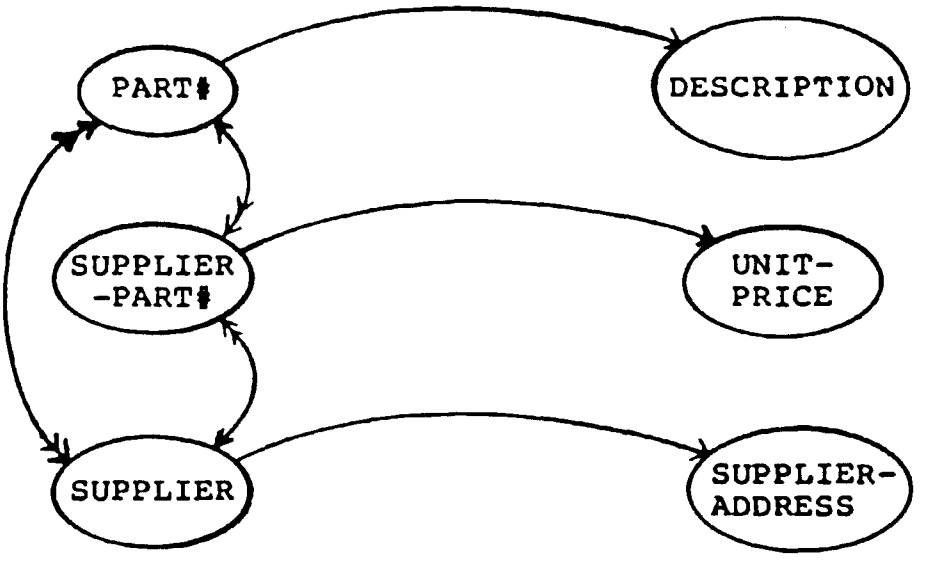


Figure 4.8 The Inventory Record in Bubble Chart Format

are redundant, since the connection between them can be derived from the concatenated eidos SUPPLIER-PART#. Usually an experienced analyst will immediately eliminate such redundant relationships. Thus, in future references to this user's view the PART#:SUPPLIER, and SUPPLIER: PART# relationships will not appear.

B. Phase II

Phase II is the process of translating the system's information requirements (as determined by Phase I) into a conceptual schema. The conceptual schema consists of:

1. eide (or data items) grouped into records (or segments)
2. records interconnected into sets (directioned relationships)

For the purposes of explanation and clarity of exposition, the conceptual schemata used in this thesis will be expressed in a graphical format. There is no reason, however, why the methodology detailed herein cannot be automated and the conceptual schemata expressed in either machine readable, or machine producible form.

Phase II consists of four stages (or steps). They are as follows:

1. create the object/attribute matrix
2. create the initial conceptual schema diagram
3. modify the schema in regard to shared attributes.
4. modify the schema to eliminate redundant sets

The first two stages are simply mechanical operations. If the Phase I results are clear-cut and unambiguous, stages 1 and 2 can be accomplished with a minimum of thought. Stages 3 and 4, on the other hand, are intuitive in nature. They are involved with modifying the initial schema; fine tuning it to fit the analyst's perceptions of the situation. These later stages give the analyst an opportunity to re-evaluate the implications of the Phase I results in light of the added perspective and clarity provided by an accurately produced information model -- the initial schema.

In the following sections a detailed discussion of each of the four stages is presented. In that discussion a simple example is built up and carried through to a final conceptual schema design. For an enhanced understanding of the Phase II methodology, however, the reader should examine Appendix C, which contains a number of other examples of this methodology.

1) Creation of the Object/Attribute Matrix

Each eidos may be used either as an object (a possessor of attributes), attribute, or both, depending upon the total information context of the system. It is the function of the object/attribute matrix to express both the character of each eidos (i.e. how it is used) and the types of relationships into which it enters. The matrix is built by first mapping one user's view of the data onto the matrix, and then step by step, adding additional views until all the users' views have been incorporated.

The object/attribute matrix is a two dimensional table. Objects are listed along the vertical axis; attributes along the horizontal. Eide which are used as both objects and attributes appear along both axes. As an example consider again the record from the parts inventory file.

PART#	SUPPLIER	SUPPLIER-ADDRESS	DESCRIPTION	UNIT-PRICE

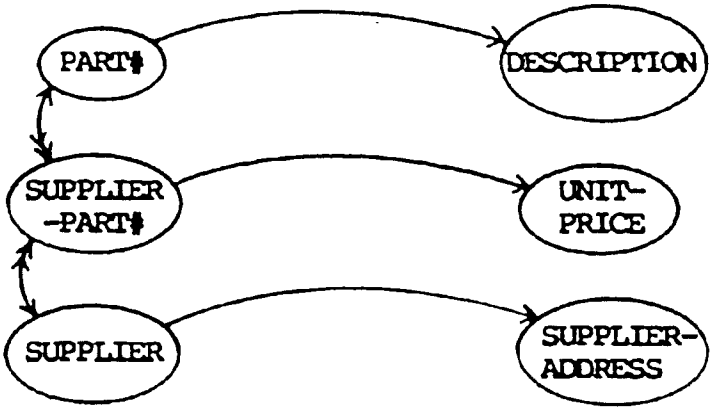


Figure 4.9 One User's View of the Inventory System

Let it be assumed that in this case parts can have multiple suppliers, suppliers multiple parts, and that price is a function of the concatenation of supplier and part. Since PART#, SUPPLIER, and SUPPLIER-PART# are objects, they would be listed along the vertical axis. PART#, SUPPLIER, SUPPLIER-ADDRESS, DESCRIPTION, UNIT-PRICE, and SUPPLIER-PART# would all be listed along the horizontal axis since all are used as attributes. Figure 4.10 shows this listing.

Attributes / Objects	PART#	SUPPLIER	SUPPLIER-ADDRESS	DESCRIP-TION	UNIT-PRICE	SUPPLIER-PART#
PART#						
SUPPLIER						
SUPPLIER-PART#						

Figure 4.10 An Incomplete Object/Attribute Matrix

Figure 4.10 is not a complete matrix however, It will be noted that although the axes have been filled in, the body of the matrix is still vacant. Whereas the axes record the identification of the eide and their function, the matrix itself expresses the types of relationships which exist. The relationships are expressed as a ratio of object to attribute (object:attribute). For example, given the assumptions listed above, the intersection of PART# and SUPPLIER-PART# would contain the value 1:M (expressing the fact that one part may have many suppliers). Likewise, the intersection of SUPPLIER and SUPPLIER-PART# would also contain the value 1:M (indicating that one supplier may supply many parts). The intersections of eide which are not directly related (such

as PART# and SUPPLIER-ADDRESS) should be left blank; as should the tautological relationship of an eidos intersecting itself.[4] Figure 4.11 is an example of a complete object/attribute matrix in the sense that it is a complete picture of the information contained in the sample record.

Attributes / Objects	PART#	SUPPLIER	SUPPLIER-ADDRESS	DESCRIP-TION	UNIT-PRICE	SUPPLIER-PART#
PART#				1:1		1:M
SUPPLIER			1:1			1:M
SUPPLIER-PART#	1:1	1:1			1:1	

Figure 4.11 An Object/Attribute Matrix of One User's View

Although most of the relationships and relationship types used in Figure 4.11 may seem obvious, the relationships regarding SUPPLIER-PART# are perhaps less so and thus require a few words of explanation. The relationship between SUPPLIER-PART# and UNIT-PRICE has been

[4]Not always will the intersection of an eidos with itself be tautological. A parts component list is an example of a situation where it would not be and where the intersection box might contain the value "1:M".

explained above (that the price of a part is a function of both the part and the supplier). Whenever a concatenation is formed, of necessity the concatenation is related to its component eide; hence the four relationships: SUPPLIER-PART:PART#, SUPPLIER-PART#:SUPPLIER, PART#: SUPPLIER-PART#, SUPPLIER:SUPPLIER-PART#.

Having mapped one user's view onto the matrix, the time has come to examine a second user's view. Suppose that the inventory system contains another file whose record is as follows:

PART#	WAREHOUSE	QTY-ON-HAND	QTY-ON-ORDER	REORDER-LEVEL

Figure 4.12 A Second Inventory Record Description

Here again the ambiguity of a file format is manifest. From the format alone, one cannot determine if a single part can be stored in multiple warehouses, and/or if a warehouse can store more than one type of part. To remedy this situation the same user's view (file) will be presented in bubble chart format. Note that here, as in the previous example, the redundant relationships (PART#:WAREHOUSE, WAREHOUSE:PART#) have been eliminated, and do not appear on the bubble chart.

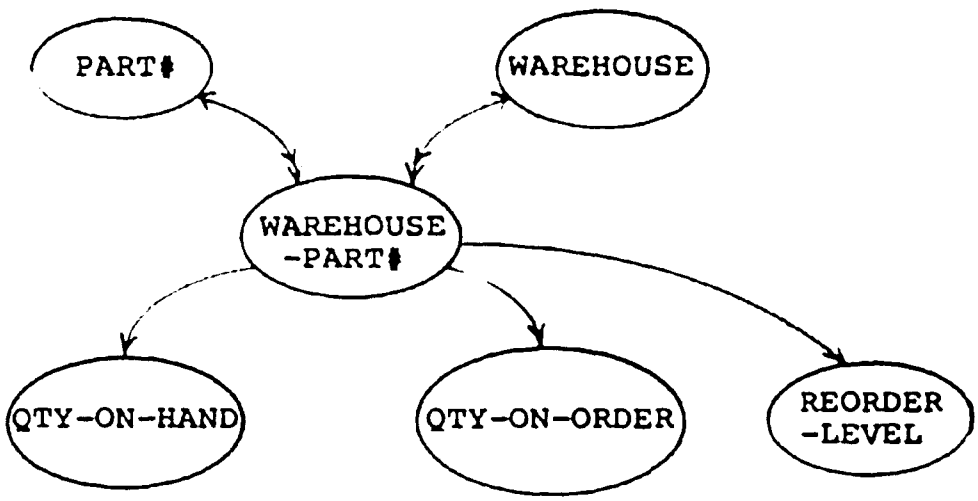


Figure 4.13 Bubble Chart of the Second User's View

The results of the inclusion of this new information into the object/attribute matrix is shown in Figure 4.14.

This process of incorporating additional users' views continues until all of the users' views are represented in the matrix. Unless a matrix contains all the users' views, it cannot be considered complete. Until it is complete the next stage of Phase II, that of creating the pure conceptual schema diagram, cannot be started.

Before turning to the discussion of this next stage, one more element of the object/attribute matrix must be illustrated.

Occasionally it will happen that two eide are related by more than one relationship. In such cases,

Objects \ Attributes	PART#	SUPPLIER	SUPPLIER-ADDRESS	DESCRIPTION	UNIT-PRICE	SUPPLIER-PART#	WAREHOUSE-PART#	WAREHOUSE	QTY-ON-HAND	QTY-ON-ORDER	REORDER-LEVEL
PART#				1:1		1:M					
SUPPLIER			1:1			1:M					
SUPPLIER-PART#	1:1	1:1			1:1						
WAREHOUSE							1:M				
WAREHOUSE-PART#	1:1							1:1	1:1	1:1	1:1

Figure 4.14 An Object/Attribute Matrix Incorporating Two Users' Views

the intersection square on the matrix is divided into as many subdivisions as there are relationships.[5] Each subdivision of the square represents one relationship. As an example of this situation consider the bubble chart of Figure 4.15, another user's view in the fictional inventory system.

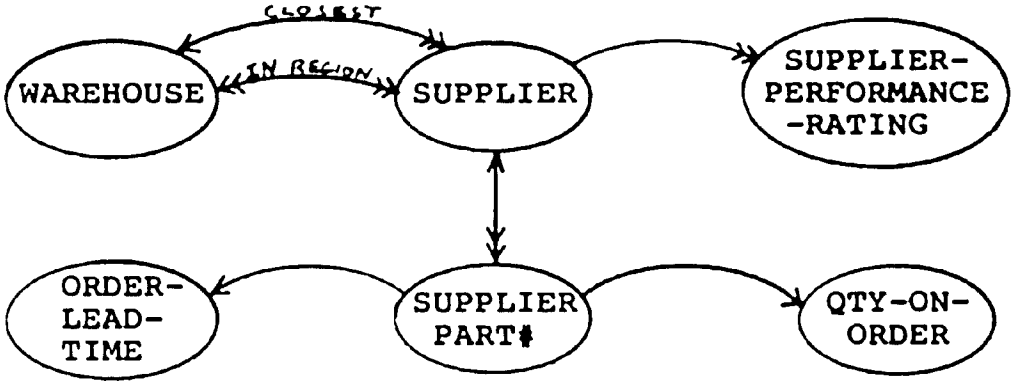


Figure 4.15 The Third User's View

In this instance there is a dual relationship from SUPPLIER to WAREHOUSE. The designers are interested in knowing not only which warehouses are in the delivery range of which suppliers; they also want to know which warehouse is the closest to a given supplier. Likewise a dual inverse relationship exists (i.e. from WAREHOUSE to SUPPLIER). The system must supply information, not

---

[5] In an automated package this problem could be handled more elegantly through the addition of another dimension to the matrix. In the manual operation however, the need for visual and graphical clarity prohibits the use of this otherwise natural third dimensional solution.

only as to which suppliers serve a certain warehouse, but also which suppliers consider a certain warehouse their prime (or closest) delivery point.

In order to illustrate how these dual relationships are mapped onto the object/attribute matrix, just this view is shown in the matrix of Figure 4.16.

Attributes / Objects	WAREHOUSE	SUPPLIER	SUPPLIER-PARTS	SUPPLIER-PERFORMANCE RATING	QTY-ON-ORDER	ORDER-LEAD-TIME
WAREHOUSE		1:M 1:M				
SUPPLIER	1:1 1:M		1:M	1:1		
SUPPLIER-PARTS		1:1			1:1	1:1

Figure 4.16 The Object/Attribute Matrix of Just the Third User's View

Note how the dual arrows of the dual relationships are mapped as split squares in the matrix. In the event that additional relationships are needed for these two eide, the intersection squares could be subdivided even further.

Having illustrated this final aspect of the object/attribute matrix, the discussion of this topic is complete. With a complete object/attribute matrix

(encompassing all the users' views) the analyst is ready to begin the process of drawing the conceptual schema diagram. In the discussions of the remaining stages, the inventory system which has been described will be regarded as a complete inventory system. The full object/attribute matrix for this sample system can be seen in Figure 4.17.

2) Creation of the Initial Schema Design Diagram

The task of this stage is to create an initial schema out of the information contained in the object/attribute matrix. A schema consists of eide grouped into records (or segments) and records interconnected in sets. Although the final schema design may have compromises and modifications built into it, the initial schema is a "pure" design (free from data redundancy) which can be produced, almost mechanically from the object/attribute matrix.

The first step in this process is the grouping of the eide into records. A record is a logical unit made up of an object and its attributes. Each object has its own record. In the non-redundant initial schema, a record should consist of one object and only those attributes which are unique to that object. Attributes which relate to more than one object should not form

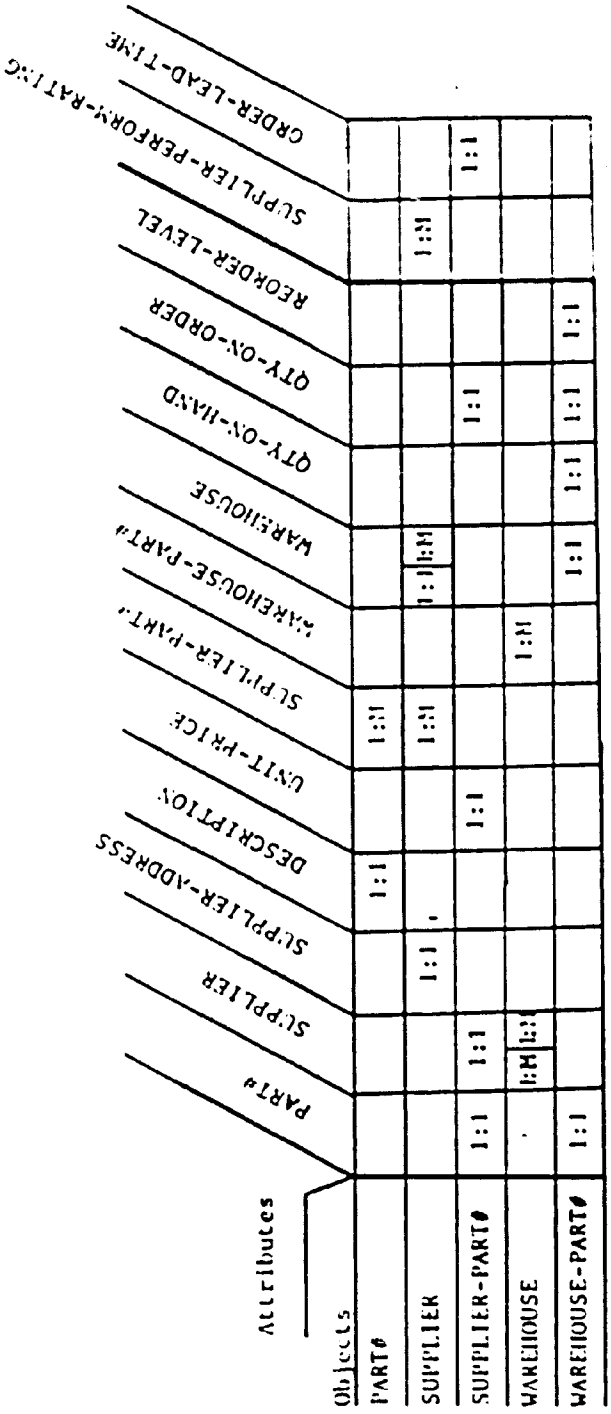


Figure 4.17 The Completed Object/Attribute Matrix

part of a record. In the inventory example, the eidos DESCRIPTION is unique to PART#. SUPPLIER-PART# could not be a member of the PART# record for two reasons: 1) SUPPLIER-PART# is an attribute of more than just PART# and 2) SUPPLIER-PART# is an object in its own right (and thus cannot be a part of another object's record).

The construction of records proceeds as follows:

a) Create a record entry for each object in the object/attribute matrix. In the inventory example this is illustrated in Figure 4.18. In that example there are five objects (PART#, SUPPLIER, SUPPLIER-PART#, WAREHOUSE, and WAREHOUSE-PART#) therefore, five record entries need to be created.

b) Identify those attributes which are eide used only as attributes. This entails examining the list of attributes and singling out those which are not also objects (i.e. a list of those eide which are pure attributes). In the example this would be: DESCRIPTION, UNIT-PRICE, QTY-ON-HAND, QTY-ON-ORDER, REORDER-LEVEL, SUPPLIER-PERFORMANCE-RATING, and ORDER-LEAD-TIME.

OBJECT:ATTRIBUTE MATRIX

OBJECTS \ ATTRIBUTES	PART#	SUPPLIER	SUPPLIER-ADDRESS	DESCRIPTION	UNIT-PRICE	SUPPLIER-PART#	WAREHOUSE-PART#	WAREHOUSE	QTY-ON-HAND	QTY-ON-ORDER	REORDER-LEVEL	SUPPLIER-PERFORMANCE-RATING	ORDER-LEAD-TIME	
PART#				1:1		1:M								
SUPPLIER			1:1			1:M		1:1	1:M			1:M		
SUPPLIER-PART#	1:1	1:1			1:1					1:1			1:1	
WAREHOUSE		1:M	1:M				1:M							
WAREHOUSE-PART#	1:1							1:1	1:1	1:1	1:1			

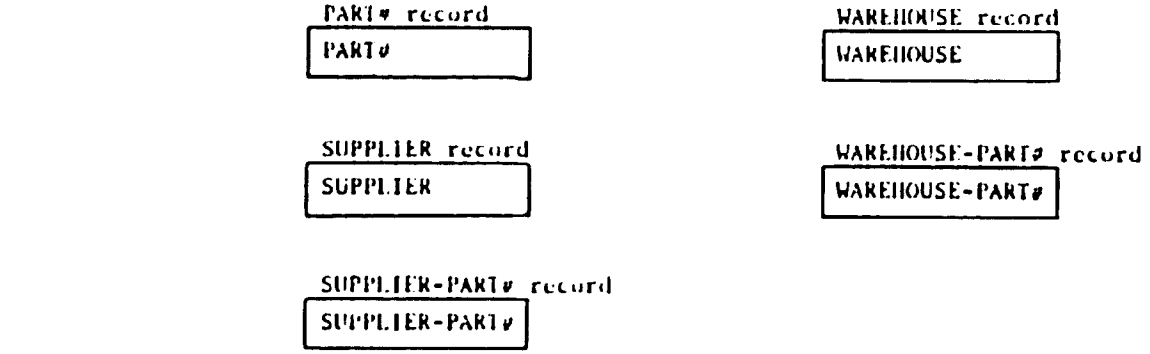


Table 4.18

Embryonic Object Records and the Object/Attribute Matrix from which they are derived

c) Add the appropriate unique attributes to each record entry. Out of the list of pure attributes, the analyst should identify those which are attributes to only one object and then add those attributes to the appropriate object's record entry. In the example all the attributes except the eidos QTY-ON-ORDER are of this ilk. Thus DESCRIPTION should be appended to the PART# record, UNIT-PRICE to the SUPPLIER-PART# record, and so on. The complete results for this step are shown in Figure 4.19.

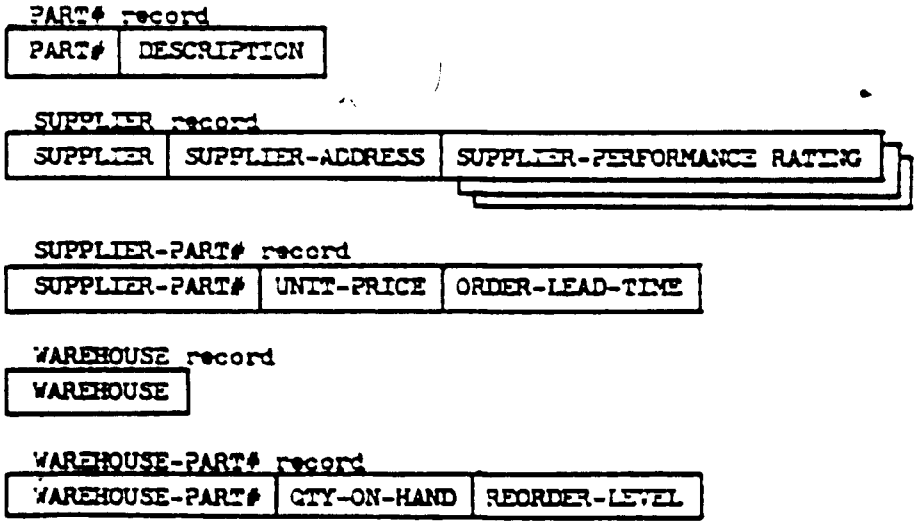


Figure 4.19 Objects and Their Unique Attributes

d) Incorporate the shared attributes into the diagram. Having entered all the objects and the unique attributes into the diagram, the only remaining eide are the shared attributes. It will be remembered that a shared attribute is an eidos which, although never used (in the system) as an object, is used as an attribute of more than one object. Shared attributes are rare occurrences. Quite often an eidos which is thought to be a shared attribute turns out, after further examination, to be either an object or a homonym for two (or more) very similar eide. In the inventory example the only shared attribute is QTY-ON-ORDER.[6] Figure 4.20 shows the state of the schema diagram after this shared attribute has been added.

---

[6]More will be said about this particular shared attribute later on.

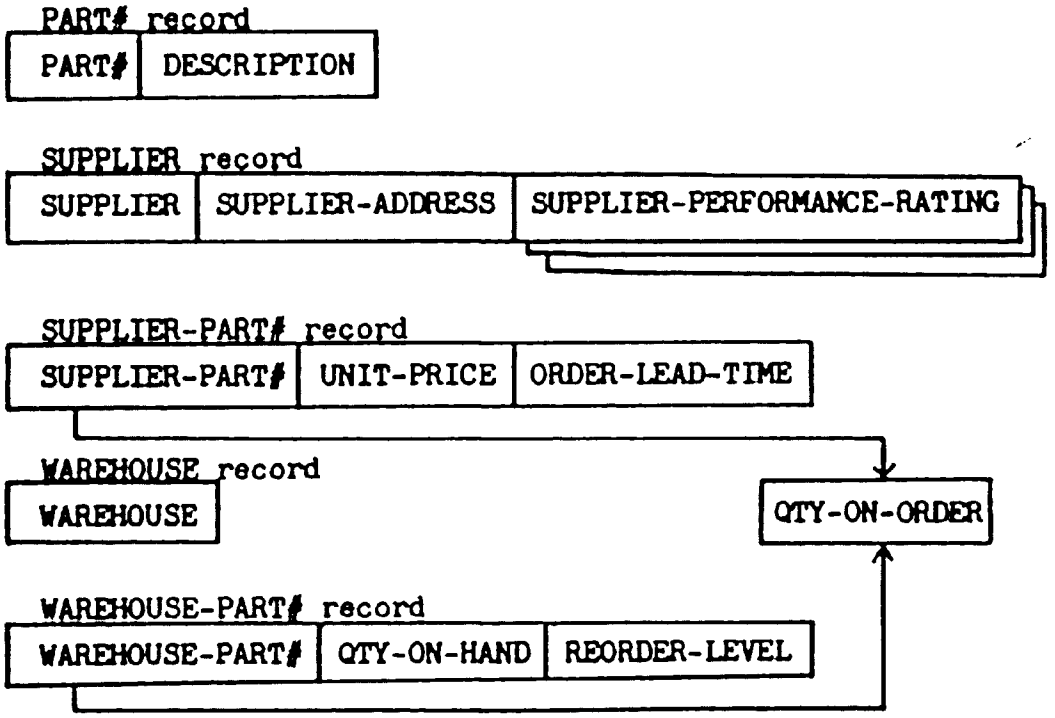


Figure 4.20 Schema Records Including Shared Attributes

At this point all the eide have been incorporated into the schema diagram. Records have been constructed which graphically express the relationship between each object and its pure attributes. The final step, then, is to complete the diagram by entering the sets (i.e. the relationships between the objects). This is probably the easiest step in the whole process. It consists of identifying those attributes which are also objects (by examining the object/attribute matrix) and drawing the appropriate type of arrow for each intersection square. In the example, if one starts from the upper left hand side of the matrix, the first attribute (which is also an object) one encounters is PART#. Proceeding down PART#'s column, the first non-vacant square is the intersection with SUPPLIER-PART#. The SUPPLIER-PART#:PART# intersection contains the value 1:1, thus a single headed arrow should be drawn from the SUPPLIER-PART# record to the PART# record (as in Figure 4.21.

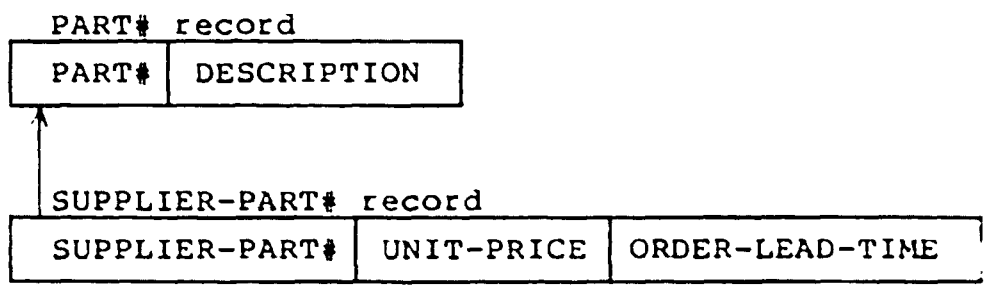


Figure 4.21 An Inter-Record Relationship

In like manner, one could continue down the PART# column to encounter the WAREHOUSE-PART# intersection. That intersection would also indicate that a single headed arrow should be drawn to PART# from WAREHOUSE-PART#.

It doesn't really matter in this examination whether one runs the columns or the rows, the end result is the same--a complete initial schema design in graphic format. For the inventory example, the complete initial schema design is shown in Figure 4.22. Note that in this figure the relative positions of some of the records differ from what they were in previous diagrams. This has been done in order to facilitate drawing a less complicated looking diagram and in no way affects the schema's information content.

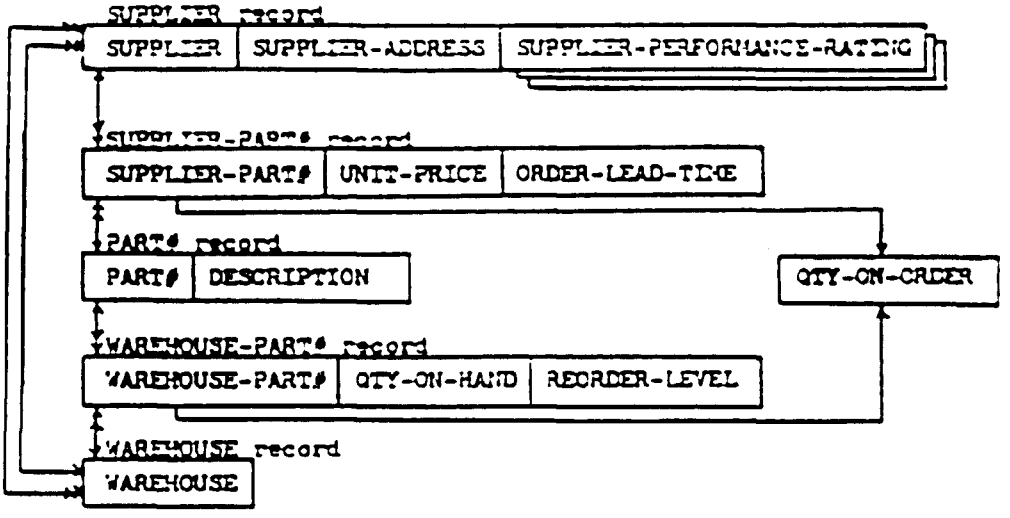


Figure 4.22 A Complete Conceptual Schema

3) Modifications to the Schema Regarding Shared Attributes

This stage is the first which requires human judgmental decisions (the designer's discretion). The problems it addresses are problems of human understanding and not shortcomings in the design generation process. Although the conceptual schema produced by stages 1 and 2 is an accurate model of the information obtained in Phase I, circumstances can exist which might necessitate modifications to that model.

Such a circumstance is the possibility that, in light of the added clarity provided by a schema model,

some of the original Phase I definitions and assumptions may demand rethinking.

The prime area of this concern is that of the shared attributes. Shared attributes are an anomaly in an otherwise clearly defined situation. Being neither unique attributes nor objects, shared attributes occupy a possibly vague middle ground. Shared attributes are rare and in general result from an insufficient analysis of the situation. What is thought to be a shared attribute is usually either an object or a homonym for similar eide.

In dealing with shared attributes, the analyst has three options:

a) Redefinition. The analyst, after reexamining the situation, may conclude that the reason a shared attribute exists is because it is ambiguously defined. On the one hand that which is thought of as being one shared attribute, might in fact be two distinct generic concepts (two distinct eide). Reviewing the inventory example, one might question whether the QTY-ON-ORDER of a particular PART# for a particular WAREHOUSE is conceptually the same idea as the QTY-ON-ORDER of a particular PART# from a particular SUPPLIER. If it is not, then instead of the single shared attribute, QTY-ON-ORDER, there should exist two non-shared attributes: WAREHOUSE-

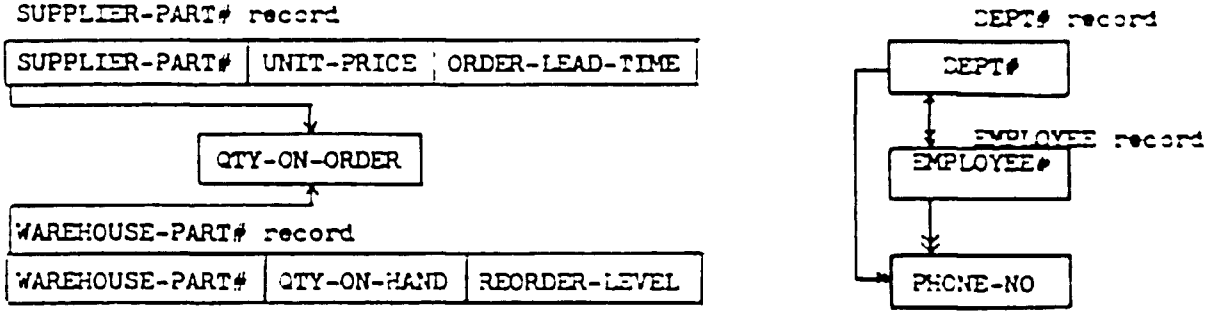
PART#-QTY-ON-ORDER and SUPPLIER-PART#-QTY-ON-ORDER (see Figure 4.23). On the other hand, that which is perceived as a single shared attribute, might advantageously be considered to be an object. The second example in Figure 4.23 illustrates such a situation. In that instance PHONE-NO is a shared attribute of both DEPT# and EMPLOYEE# (enabling the system to report the phone number for any given department and/or employee). It might, however, be useful to be able to ascertain to whom a particular phone number belongs (i.e. establish PHONE-NO as an object with attributes DEPT# and EMPLOYEE#). In either of these two situations all that is involved is a simple redefinition of function. Occasionally a more extreme re-evaluation is necessary but that will be discussed in (c) below.

b) Maintenance of the status quo. One cannot rule out, a priori, the possibility that a shared attribute is really the most accurate and most comprehensible description of the situation being modeled. If such is the case, then there is no reason to remove the shared attribute from the schema and it should, most certainly, be retained.

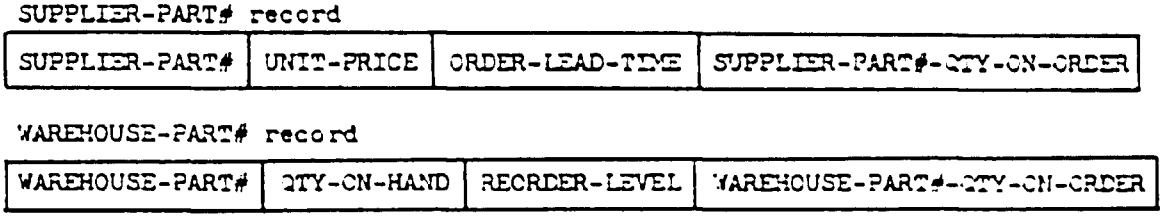
c) Re-evaluation of Phase I assumptions. This is the third and most drastic option available. It calls for the analyst to rethink and re-examine some of the

OPTION A

Original Situations



After Redefinition



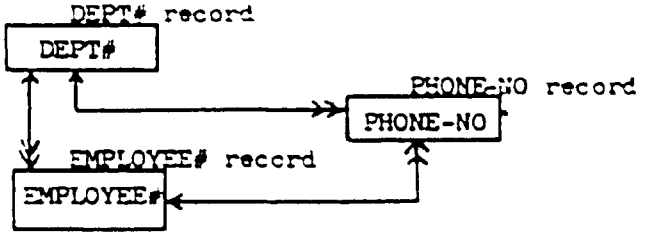


Figure 4.23 Redefinition of Shared Attributes

basic Phase I assumptions. In the inventory example this re-examination might yield the following analysis.

- QTY-ON-ORDER as determined by Phase I is ambiguous and needs re-evaluation. As it stands now, QTY-ON-ORDER, can refer to the quantity of a PART# on order from a particular SUPPLIER or, the quantity of a PART# on order for a particular WAREHOUSE. The logical view of wanting to know how many of a certain PART# are on order for a particular WAREHOUSE from a particular SUPPLIER (see Figure 4.24) is not accounted for.

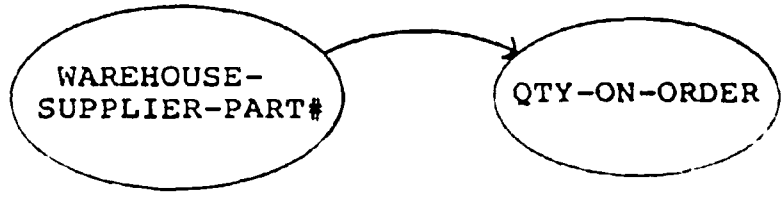


Figure 4.24 A Hitherto Unaccounted-for View

- All three views of QTY-ON-ORDER (as an attribute of SUPPLIER-PART#, as an attribute of WAREHOUSE-PART# and as an attribute of WAREHOUSE-SUPPLIER-PART#) are really derivable quantities. In each case QTY-ON-ORDER can be obtained by summing the number of PARTS listed on each of the appropriate outstanding orders. An ordered quantity of parts is actually an attribute of a specific order. The quantity of parts on order is therefore, a function of the concatenation of PART# and ORDER#. Although the eide ORDER# and ORDER#-PART# are not currently features of the system, they are basic concepts whose inclusion would prove beneficial.

- All the mentioned relationship requirements can be met using the following user's view (Figure 4.25).

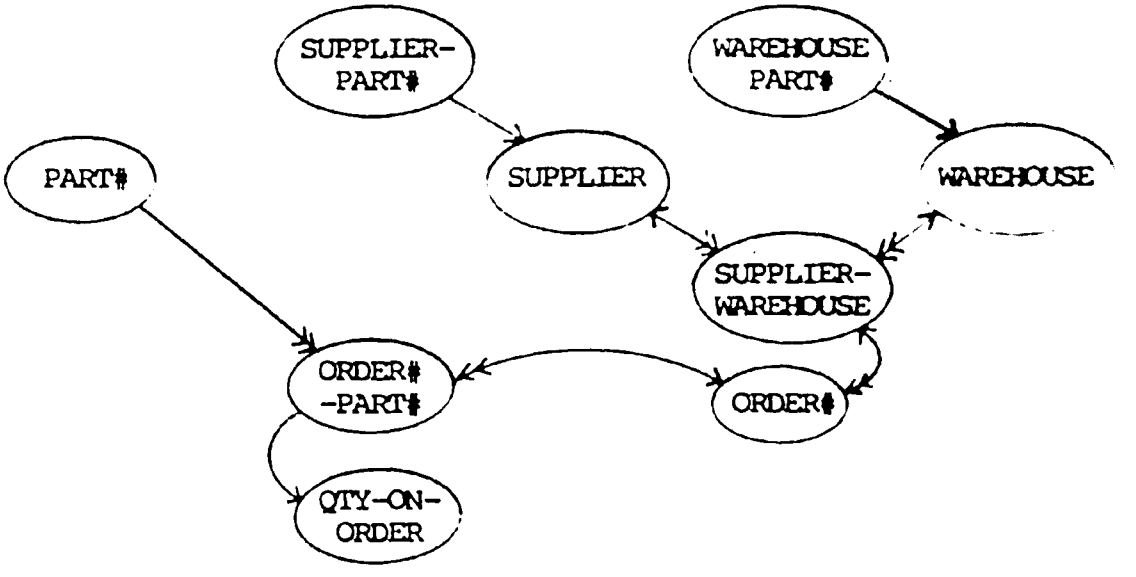


Figure 4.25 A More Accurate View of the Role of QTY-ON-ORDER

- The incorporation of this analysis into the overall picture renders the previous usage of QTY-ON-ORDER (that of an attribute of both WAREHOUSE-PART# and SUPPLIER-PART#) a derivable quantity. As such it is a source of implicit redundancy and should be removed from the schema (thus also removing the necessity of renaming it).

The decision as to which option to choose must be a human, intuitive choice. The analyst can only examine each option in the light of his own knowledge and experience. In the inventory example any of the three options could be a viable alternative, but it is the author's opinion that in this case, the third option (the re-evaluation) holds the most promise. The

revisions which such a re-evaluation would necessitate (to both the object/attribute matrix and the schema diagram) are shown in Figures 4.26 and 4.27.

4) Modifications to the schema to eliminate redundant sets

In incorporating the various users' views into the schema, it often happens that certain sets are rendered redundant. A set is redundant when the relationship it expresses can be deduced from other non-redundant sets. The classical example of a redundant set occurs most naturally in a hierarchical situation.

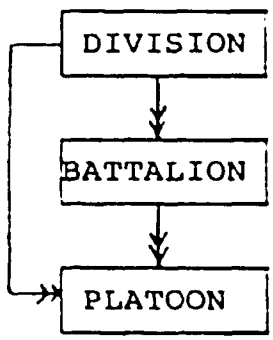


Figure 4.28 A Classic Example of a Hierarchy

In Figure 4.28 the relationships between DIVISION and PLATOON are directly deducible from the relationships between DIVISION and BATTALION and the relationships between BATTALION and PLATOON. Thus the DIVISION: PLATOON and the PLATOON:DIVISION relationships are

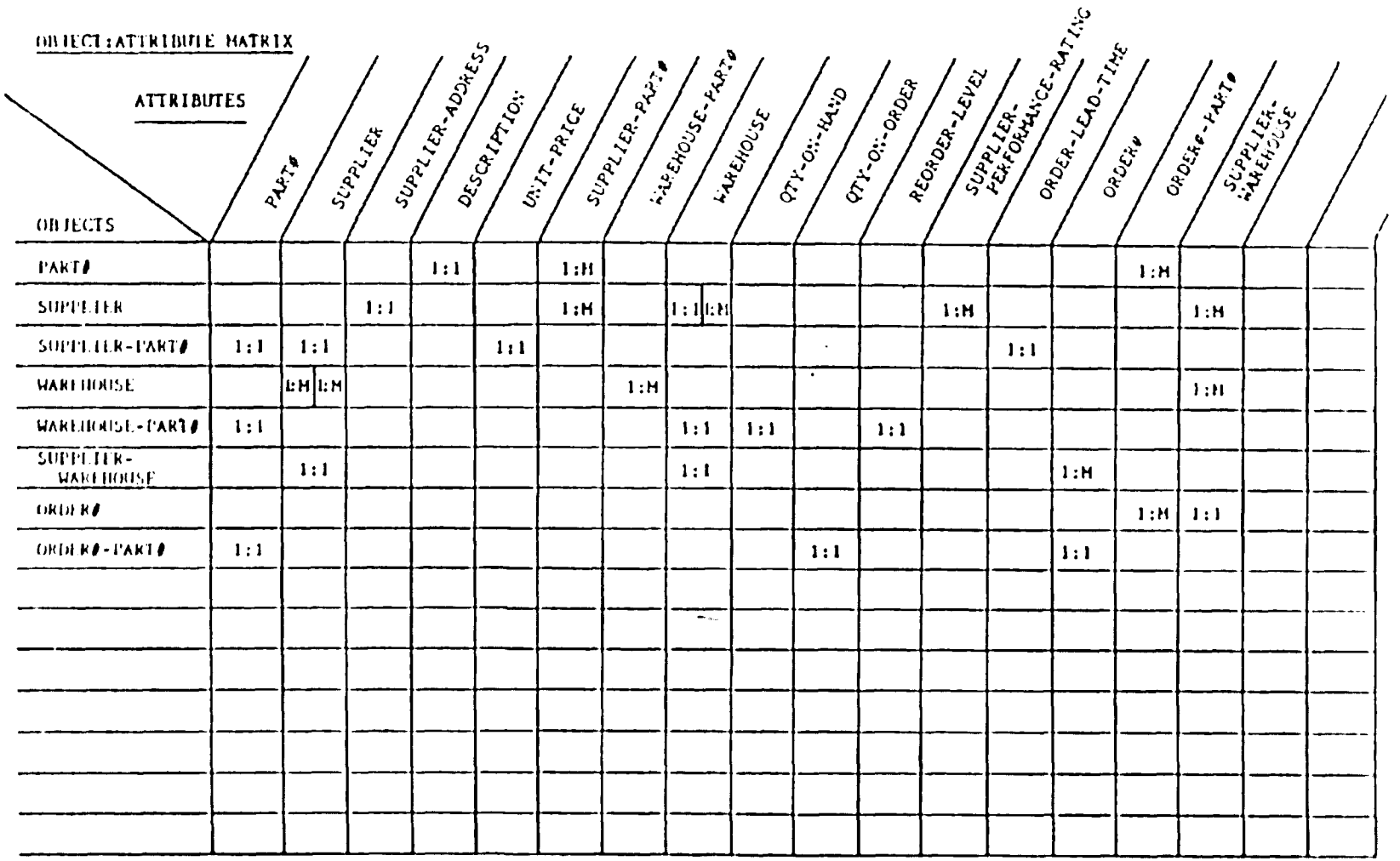


Figure 4.26 The Revised Object/Attribute Matrix

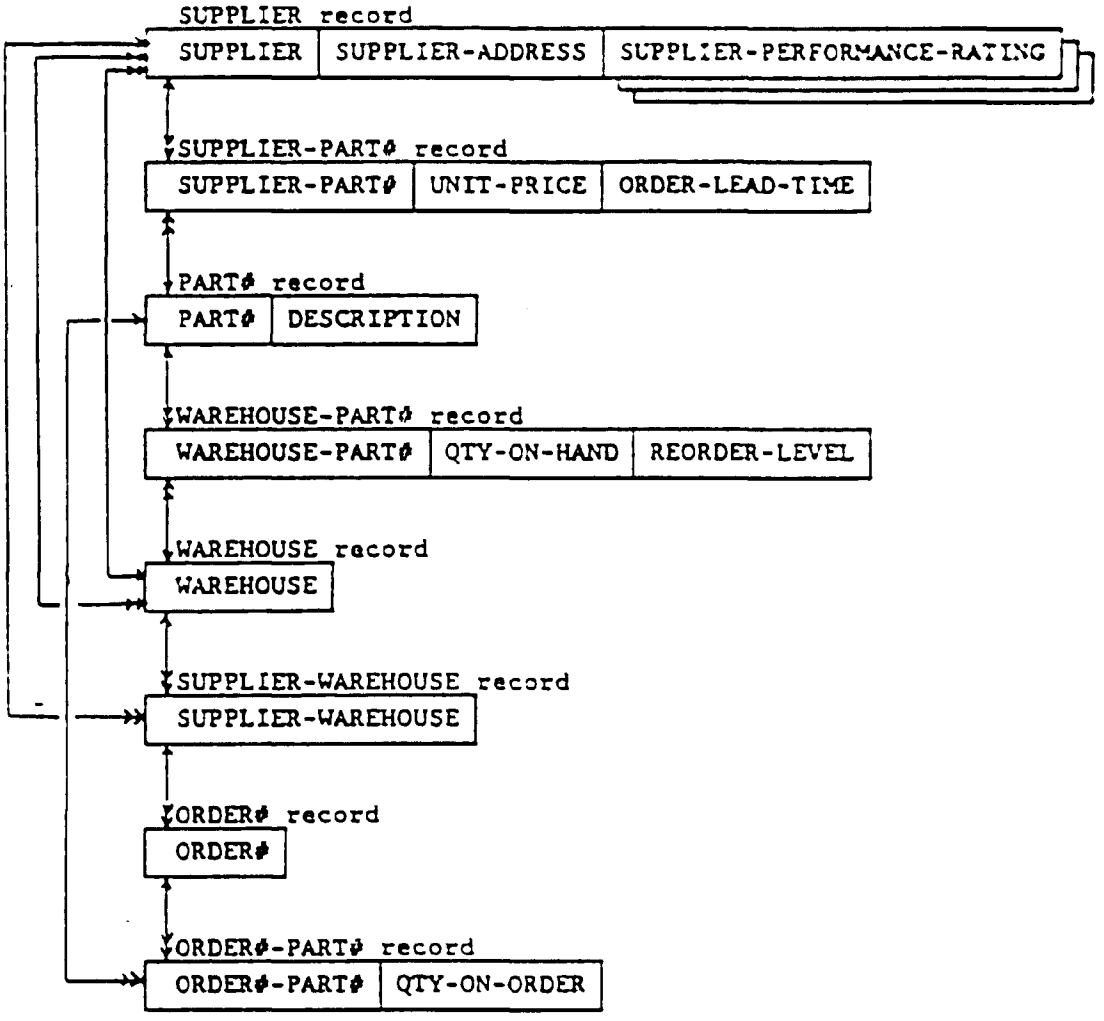


Figure 4.27 The Revised Conceptual Schema

redundant.

In most circumstances it is wise to eliminate redundant sets (relationships). They add little conceptual content to the model and tend to clutter the picture. Occasionally, however, an analyst will choose to retain a redundant relationship in order to emphasize the existence of a very important relationship (one which should be explicitly expressed and not implied).

Although, as was stated in the discussion of Phase I, an experienced analyst can usually spot and weed out redundant relationships before they are incorporated into the schema, great care must be exercised in the removal of redundant relationships. Not all relationships which appear to be redundant are really redundant. Consider the two examples in Figure 4.29. The diagrams for the two situations are identical. The DIVISION to PLATOON, PLATOON to DIVISION relationships are truly redundant. The DEPT# and PHONE-NO relationships need not be. If the list of phone numbers includes the home numbers of the employees and/or the department has an official line (or lines) not assigned to any employee, then the DEPT# to PHONE-NO and PHONE-NO to DEPT# relationships are definitely not redundant.

In the inventory example all set redundancies were carefully weeded out of each user's view before

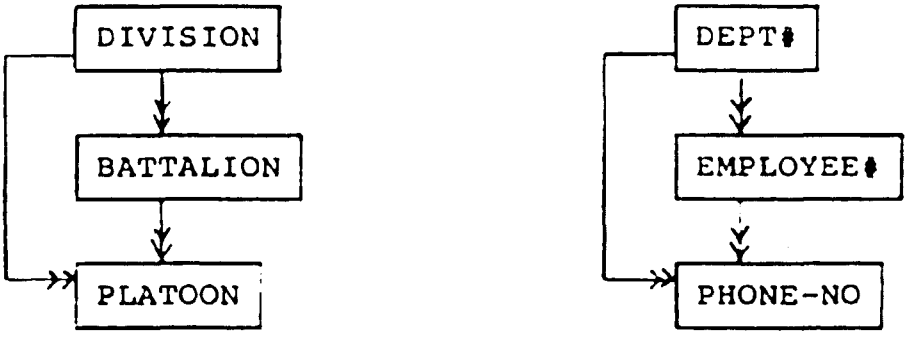


Figure 4.29 Redundant and Non-Redundant Sets

any information was incorporated into the object/ attribute matrix. Nevertheless, as the schema currently stands, set redundancy does exist. Figure 4.30 illustrates this redundancy. The set connecting a SUPPLIER to all the WAREHOUSEs it serves, and the set connecting a WAREHOUSE to all the SUPPLIERs which serve it, are no longer necessary (as this information is now obtainable through the concatenated eidos SUPPLIER-WAREHOUSE).

The cause of this redundancy stems not from having overlooked a redundant set within a particular user's view, but rather from an aspect of one user's view (in this case the concatenated eidos SUPPLIER-WAREHOUSE) rendering redundant a set in another user's view (the SUPPLIERS-IN-REGION, and WAREHOUSES-IN-REGION sets). Typically an analyst will work on only one user's view at a time, thus in stages 1 and 2 it is very difficult

to guard against this type of redundancy. Given the overall view provided by a schema diagram however, all such redundancies are discernible.

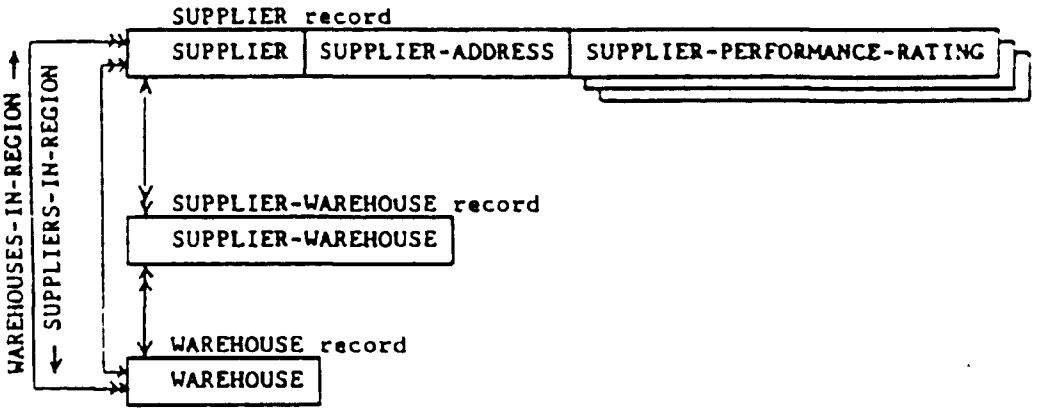


Figure 4.30 Two Redundant Sets in the Inventory Schema

In the inventory example there is no special requirement to retain the redundant sets and they should be eliminated. This final modification completes the inventory example and the resultant conceptual schema is shown in Figure 4.31.

C. Concluding Remarks on the Conceptual Schema Design

The goal of this methodology has been to place the burden of human endeavor where it belongs -- on the design and understanding of information systems, not on

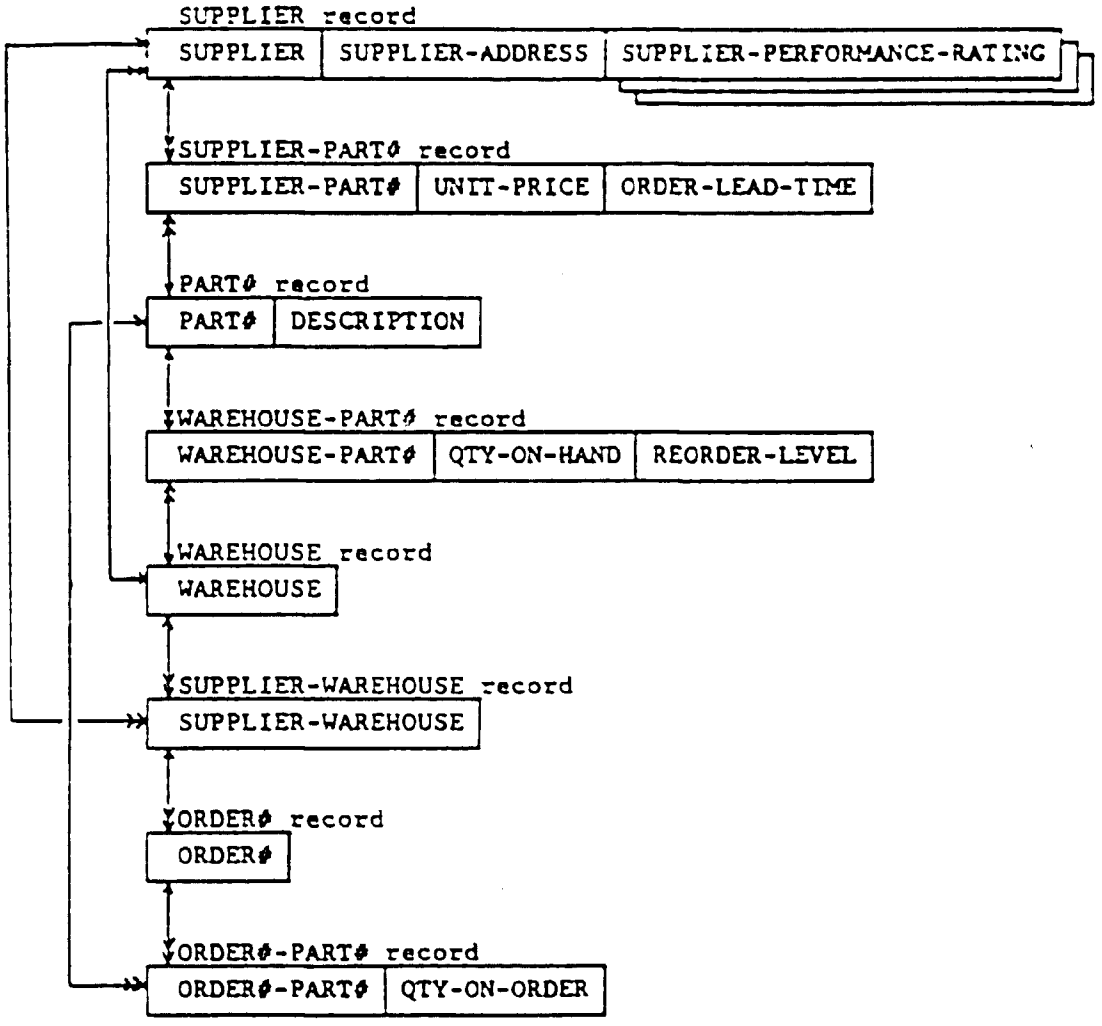


Figure 4.31 The Final Conceptual Schema

the mechanics of a schema design. The complete design process is an iterative one. Initially the analyst collates the various "users' views" of the system (Phase I). These initial perceptions are entered into a model generator (stages 1 and 2 of Phase II) which produces an accurate model, the initial schema. In light of the enhanced clarity which the model provides (an overall view of the information system rather than a series of private "users' views"), the analyst can re-evaluate some of the initial perceptions and modify the model accordingly (stages 3 and 4 of Phase II). The cycle of re-evaluation and modification can then continue until the analyst is satisfied that the schema is an accurate and correct model.

The ease in which schemata are produced from a set of users' views (analysts perceptions) renders the Database very simple to update (modify). The analyst need only be concerned with obtaining an accurate understanding of the new requirements in order to change an existing schema, since the actual process of producing the schema is mechanical. Insofar as most schemata are not static constructs (most information systems being in a constant state of flux), this is an important feature.

The result of the methodology detailed above is a conceptual schema. It is a comprehensible, and

accurate, simple to design reflection of a system's information needs. It also purports to be a solid foundation for the construction of a practical working schema. The manner in which this transformation is accomplished is briefly described in the next section.

IV. From Conceptual Schema to Working Schema

A conceptual schema is a tool, a design aid. Its strength lies in its ability to be a flexible, independent and accurate model of a system's information requirements. In the conceptual, unrestricted form, this tool is not an operational tool (not a working schema). One of the purposes of designing a conceptual schema, however, is to create a base from which a working schema may easily be derived.

The transformation from a conceptual schema to a working schema should be a simple one. It requires merely that the idiosyncratic restrictions of a specific DBMS package and of a specific machine environment be imposed upon the unrestricted conceptual model. These restrictions are in the nature of compromises to the pure unrestricted design. Although in order to delve into specific details would require a thorough acquaintance with a specific environment (and is thus beyond the scope of this thesis), examples can be presented of the type of compromises an analyst might be required to make.

The conceptual schema is so close to CODASYL DBTG schema format, that little or no effort is required to convert a conceptual schema to any of the CODASYL DBTG based packages. Nevertheless, CODASYL as well as most other DBMS packages does not support M:M relationships;[7] a feature which can be found in a conceptual schema. The removal of a M:M relationship is not a complex operation. All that it requires is the creation of a new record (a nub) which consists of the concatenation of the two eide involved in the original M:M relationships. Figure 4.32 illustrates this M:M removal operation.

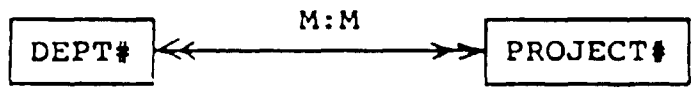
When one begins to consider machine environment factors, one invariably encounters the ubiquitous "space/time" tradeoff (response times can be improved at the cost of additional storage space and requisite storage space can be reduced at the cost of response time).

One way response times can be improved is by introducing additional data and set redundancy into the Database. The more places a specific piece of data exists, the less searching the system must do.

---

[7]What in CODASYL terminology is called an M:M relationship and which this author has described as being <u>two</u> complementary 1:M relationships.

The M:M relationship (Each department can work on many projects and each project can be apportioned among many departments)



The adapted relationship (each of the original eide now has an 1:M relationship with the concatenation)

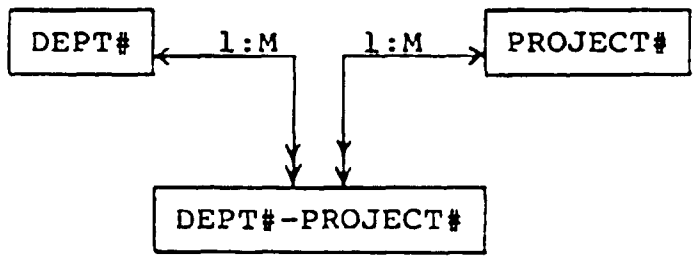


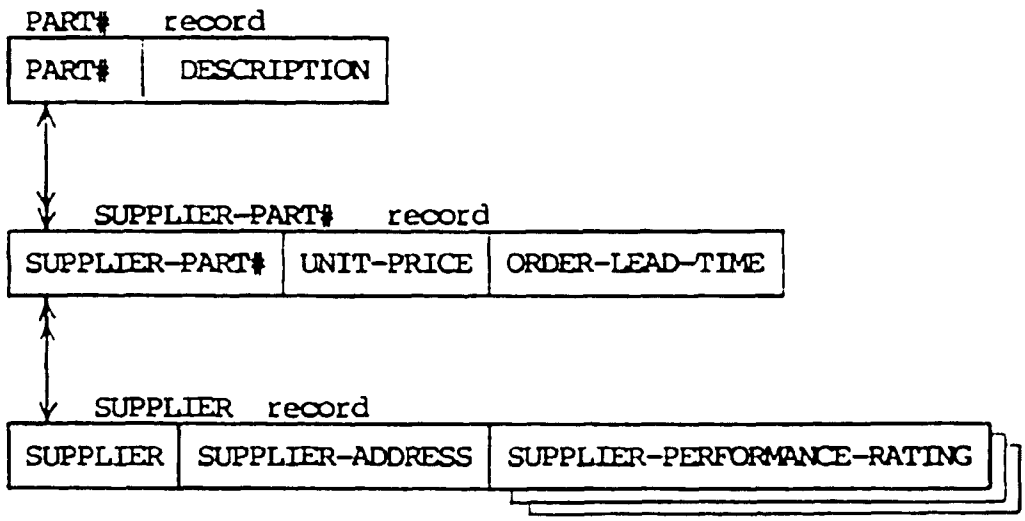
Figure 4.32 Elimination of M:M Relationships

The more sets a specific record belongs to, the quicker it can be found. Both additional sets and data require additional storage. How much storage cost is the extra speed worth?

If storage (rather than response time) is at a premium then storage costs can also be cut down. Certainly the removal of all data (and set) redundancy is one feasible option. Reducing to a minimum such niceties as hashed key accesses is another. A not-so-obvious source of elimination of data redundancy is the concatenated eide. Often the concatenation itself can be

eliminated from its own record. Figure 4.33 illustrates how this can be done.

With the concatenated eidos SUPPLIER-PART#



Without the concatenated eidos SUPPLIER-PART# (All accesses to the SUPPLIER-PART# record must access both parent records in order to identify the particular record being accessed.)

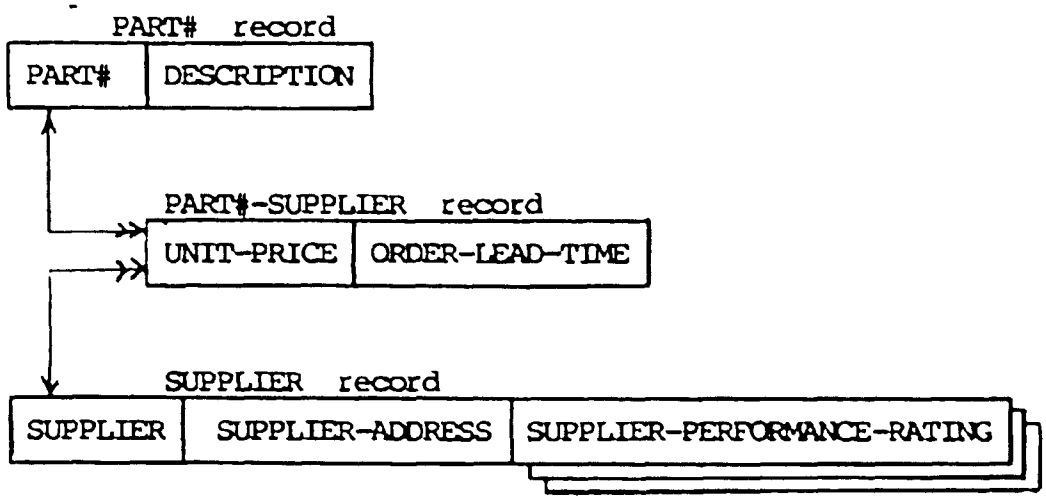


Figure 4.33
Removing Redundant Data From Concatenated
Keys at a Cost to Access Time and Complexity

Depending upon one's own needs and constraints, the conceptual schema can be bent and reshaped until there is barely a hint left of its original form. Each change, however, is a compromise, an introduction of pollution into the pure information model. Despite the simplicity of incorporating compromises into a conceptual schema, the analyst must never lose the long range perspective. The true information model is the conceptual schema; compromises to it are to be eschewed wherever possible.

V. Summary

The subject of Database is information. It is the end for which the Database exists. Information is a very practical and applicable commodity. Information, in order to be useful, must be accessible and it is the function of a Database to make information accessible. This task is a difficult one, abounding in complexities. A Database, by virtue of the end it serves is not a simple structure; nevertheless, despite a common belief to the contrary, Database design need not be overly complex.

There are two major aspects to Database design: the creation of a pure information model and the modification of the pure model to conform to the exigencies of

the current state of technology. The latter aspect, the incorporation of environment dictated compromises, is a complex operation. Its complexity is directly proportional to the complexity and the restrictiveness of the given DBMS package being used. The primary aspect however, the design of the conceptual schema, need no longer be viewed as a complex operation. The methodology which this thesis has presented, demonstrates that schema design can be a mechanical process. If complexity does exist in this aspect of Database design, it stems from the complexity of the system being modeled, not from the Database design process itself.

As any experienced analyst knows, information systems can be exceedingly complex and challenging. If the tools the analyst has to work with are also complex and challenging, the problems are unnecessarily multiplied. By removing the complexity from one aspect of the analysts' Database tool, this author hopes to have helped clear the path to unencumbered confrontations with the proper problems of systems analysis.

CHAPTER FIVE

SUGGESTIONS FOR FURTHER RESEARCH

I. An Automated Design Methodology

Throughout Chapter Four's explanation of the proposed methodology, numerous references were made to the methodology's "mechanical" aspects (specifically stages 1 and 2 of Phase II). Obviously the term "mechanical" was used as a metaphor, denoting that the activities involved could be accomplished by a machine (i.e. a computer), not that the methodology itself is in machine readable form. By shifting the emphasis a little (and noting that every piece of drudgery which can be accomplished by a machine should be accomplished by a machine) it can be asserted that those aspects of this the methodology which are automatable, should be automated.

The result of such an automation effort should produce a useful and worthwhile Database Design Package. The most powerful design would be an interactive (conversational) program which would encompass all of the Phase II stages. The overall program logic of a Phase II package might be similar to the following:

1. Read in the initial Phase I results.
2. Create the object/attribute matrix.

3. Create the initial schema design (from the object/attribute matrix).

4. Output the schema design.

5. Output comments which would indicate possible points where the initial schema might be modified (i.e. shared attributes, data redundancies, set redundancies etc.).

6. Request modifications; if none are needed end the program.

7. Incorporate the modifications into the object/attribute matrix and into the schema.

8. Return to step #4.

Although the little logic list above is in no way exhaustive, it suffices to illustrate that the automation of the proposed Database design methodology is not an impossible undertaking. Considering the benefits such an automated package could provide, this seems a most worthy enterprise.

II. A Definitive and Systematic Approach to Phase I

In the discussion of Phase I (identification of the information to be modeled) it was stated that "There are many methods to accomplish this task [Phase I] and many tools to aid the analyst in the endeavor." This is not to say that all the methods are of equal value or that even any one method can be considered definitive. The systematic gathering and modeling of a system's descriptions is a problem with applications far byond the realm of just Databases.

This thesis has presented the minimum "information requirements" that an "information identification" effort would need for Database design. A researcher might limit his initial quest to devising a systematic way of meeting those requirements and then expand to include a wider horizon. Even if the scope were limited to only certain types of applications, a definitive, systematic approach to the identification (and notation) of systems' information requirements would be a significant boon to the DP industry.

III. A Test of the Methodology

Centrally crucial to the total endeavor of this thesis is the idea that the methodology produced must be applicable in real world situations. Throughout the analysis and discussion there has been an emphasis on comprehensibility, ease of use, and simplicity; practicality has been a principle criteria ("the touchstone of pragmatism") by which other methodologies have been judged. In the light of such a goal it is imperative that the proposed methodology be tried and tested in a "real world" environment.

Ideally a true test must consist of a trial implementation of the methodology by "front line analysts" working in various DP environments. The methodology, is after all, meant as a Database design tool (i.e. a tool

for DP practitioners). The front line Database analysts should therefore be its judges.

A test of this sort would have analysts work for a period of time with the proposed methodology. It would then require feedback from the analysts in the form of evaluations. These could take the form of questions similar to the following:

1. Which features of the methodology proved convenient?
2. Which features of the methodology proved annoying?
3. Were the schema designs flexible and easy to work with?
4. Could the conceptual designs be easily modified to meet package and machine restrictions?
5. Were the Databases produced functionally viable?
6. Were there any special problems encountered?
7. How does this methodology compare with others with which you are familiar?

This test would serve two very important functions. Certainly, as with any valid test, it could be used to judge the effectiveness and the value of the proposed methodology. In addition, however, a test of this kind could be used to fine tune the methodology; to make it more responsive to the needs of the practitioners. The desired result, is after all, to produce

a product which is useful, toward that end a test would provide invaluable feedback.

If this proposed methodology is not to be just another adorning flower upon an ivory tower, then a project in line with this suggestion must eventually come into being. All ideas and methodologies can be improved upon. Improvement is the natural direction of growth. This suggested research would provide the requisite feedback for that growth, and as such can be considered an essential component in the creation of this methodology.

BIBLIOGRAPHY

Books

Chen, Peter Pin-Shan. The Entity-Relationship Approach to Logical Data Base Design. The Q.E.D. Monograph Series Data Base Management, No. 6. Wellesley, Massachusetts: Q.E.D. Information Sciences, 1977a.

Date, C. J. An Introduction to Database Systems. Reading, Massachusetts: Addison-Wesley, 1977 (2nd ed.).

Jones, Paul E., Jr., Data Base Design Methodology: A Logical Framework. The Q.E.D. Monograph Series Data Base Management, No. 3. Wellesley, Massachusetts: Q.E.D. Information Sciences, 1976.

Katzen, Harry Jr. Computer Data Management and Data Base Technology. New York: Van Nostrand Reinhold Company, 1975.

Lyon, John K. An Introduction to Data Base Design. New York: John Wiley & Sons, 1971.

Lyon, John K. The Database Administrator. New York: John Wiley & Sons, 1976.

Martin, James. Principles of Data-Base Management. Englewood Cliffs, New Jersey: Prentice-Hall, 1976.

Martin, James. Computer Data-Base Organization. Englewood Cliffs, N.J.: Prentice-Hall, 1977 (2nd ed.).

Meadows, Charles T. Applied Data Management. New York: John Wiley & Sons, 1976.

Ross, Ronald G. Data Base Systems Design, Implementation, and Management. New York: Amacom, 1978.

Sundgren, Bo. Theory of Data Bases. New York: Petrocelli/Charter, 1975.

Wiederhold, Gio. Database Design. New York: McGraw-Hill, 1977.

Yourdon, Edward. Design of On-Line Computer Systems. Englewood Cliffs, N. J.: Prentice-Hall, 1972.

Articles and Conference Proceedings

Ashany, R. and M. Adamowicz. "Data Base Systems." IBM Systems Journal, 15, No. 3 (1976), pp. 253-263.

Bernstein, Philip. "Synthesizing Third Normal Form Relations from Functional Dependencies." ACM Transactions on Database System, 1, No. 4 (1976), pp. 277-298.

Blasgen, M. W. and E. K. P. Eswaran. "Storage and Access in Relational Data Bases." IBM Systems Journal,16, No. 4 (1977), pp. 363-377.

Canning, Richard G. "Creating the Corporate Data Base." EDP Analyzer, 8, No. 2 (1970).

Canning, Richard G. "Organizing the Corporate Data Base." EDP Analyzer, 8, No. 3 (1970).

Canning, Richard G. "Processing the Corporate Data Base." EDP Analyzer, 8, No. 4 (1970).

Canning, Richard G. "Data Security in the CDB." EDP Analyzer, 8, No. 5 (1970).

Cardenas, Alfonso F. "Evaluation and Selection of File Organization - A Model and System." Communications of the ACM, 16, No. 9 (1973), pp. 540-548.

Chen, Peter Pin-Shan. "The Entity-Relationship Model - Towards a Unified View of Data." ACM Transactions on Database Systems, Vol. I, No. 1, March 1976, pp. 9-36.

Chen, Peter Pin-Shan. "The Entity-Relationship Model - A Basis for the Enterprise View of Data." AFIPS Conference Proceedings, 1977 National Computer Conference. 13-16 June 1977. Montvale, N. J.: AFIPS Press, 1977b, pp. 77-84.

Durchholz, R. and G. Richter. "Concepts for Data Base Management Systems." Data Base Management, IFIP Working Conference on Data Base Management. 1-5 April, 1974. Amsterdam: North Holland Publishing Co., 1974, pp. 97-122.

Gerritsen, Rob. "A Preliminary System for the Design of DBTG Data Structures." Communications of the ACM, 18, No. 10 (1975), pp. 551-557.

Grotenhuis, F. J. W. "STAF: Standard Automation Fundaments, A Model for Automatic Processing." Data Base Management, IFIP Working Conference on Data Base Management. 1-5 April, 1974, Amsterdam: North Holland Publishing Co., 1974, pp. 313-335.

Finneran, Thomas R., J. Shirley Henry. "Structured Analysis for Data Base Design." Datamation, November 1977, pp. 99-113.

Heyne, G. F. and C. J. Daniel. "Design Techniques for a User Controlled DB/DC System." IBM Systems Journal, 16, No. 4 (1977), pp. 344-362.

Kissinger, Henry. "White Revolutionary: Reflections on Bismarck." Daedalus, 97, No. 3 (1968), pp. 888-924.

Nijssen, C. M. "Data Structuring in the DDL and Relational Data Model." Data Base Management, IFIP Working Conference on Data Base Management. 1-5 April, 1974. Amsterdam: North Holland Publishing Co., 1974, pp. 363-384.

Nunamaker, J. F., Jr., Ben R. Konsynski, Jr., Thomas Ho, and Carl Singer. "Computer-Aided Analysis and Design of Information Systems." Communications of the ACM, 19, No. 12 (1976), pp. 674-687.

Raver, N. and G. U. Hubbard. "The Automated Logical Data Base Design: Concepts and Applications." IBM Systems Journal, 16, No. 3 (1977), pp. 287-312.

Senko, M. E. "Data Structures and Data Accessing in Data Base Systems Past, Present, Future." IBM Systems Journal, 16, No. 3 (1977), pp. 208-257.

Sibley, Edgar H. and Larry Kerschberg. "Data Architecture and Data Model Considerations." AFIPS Conference Procedures, 1977 National Computer Conference. 13-16 June 1977. Montvale, N. J.: AFIPS Press, 1977, pp. 85-96.

Strocker, P. M., and P. A. Dearnly. "A Self-Organizing Data Base Management System." Data Base Management, IFIP Working Conference on Data Base Management. 1-5 April, 1974, Amsterdam: North Holland Publishing Co., 1974, pp. 334-349.

Sundgren, Bo. "Conceptual Foundations of the Infological Approaches to Data Bases." Data Base Management, IFIP Working Conference on Data Base Management. 1-5 April, 1974. Amsterdam: North Holland Publishing Co., 1974, pp. 61-96.

Titman, P. J. "An Experimental Data Base System Using Binary Relations." Data Base Management, IFIP Working Conference on Data Base Management. 1-5 April, 1974. Amsterdam: North Holland Publishing Co., 1974, pp. 351-361.

Towner, L. E. "Non-Codasyl DBMS - A Bad Choice for Users." Computerworld, 6 Feb. 1978, p. 21.

Wood, Roy, and Robert B. Chamberlain. "Once for Designers, DBMS Now Keyed to User." Computerworld, 6 Feb. 1978, p. 26.

Reports and Manuals

Bethlehem Steel Corp. Data Processing Services. IMS Standards Manual User's Guide (rev. 1), Bethlehem, Pennsylvania: 1 June, 1977.

Committee on Data Systems Languages (CODASYL). Data Base Task Group Report April 1971. New York: Association for Computing Machinery, 1971.

Digital Equipment Corp. Data Base Management System Administrators Procedures Manual. D.E.C. Maynard, Massachusetts: 1977.

IBM Corp. Data Base Design Aid Version 2 General Information Manual. IBM. White Plains, New York: 1977 (GH20-1626-2).

Unpublished Working Papers

Mitoma, Michael, F., Optimal Data Base Schema Design. Ann Arbor: Michigan University, 1975.

Weldon, Jay-Louise. "A Data Base Configuration Model." New York: New York University, Aug. 1977.

Weldon, Jay-Louise. "Using Database Abstractions for Logical Design - A Case Study." New York: New York University, Oct. 1977.

APPENDIX A

AN IN-DEPTH PROBLEM SCENARIO

I. Design and Traditional Files

Before turning attention to Database design techniques it behooves one to establish some perspective as to the origins and evolution of the Database concept. This requires some understanding of the design environment previous to Database technology. It was, after all, an environment of narrow horizons, demanding of designers only that they satisfy the needs of a given particular application, or at most, the needs of a single specific system.

In the first DP systems the overwhelming number of applications were batch oriented. Hardware limitations and storage costs combined to restrict most files to a sequential storage medium (magnetic tapes and/or perforated paper). This meant that most file processing had to be sequential, but that the potential length (the number of records) and potential breadth (the record size) of files could be considered (within reason) unbounded.

Information was considered parochial in nature; each system (all files and programs) being the private fiefdom of its owner. System sovereignty was a well

respected right which when transgressed required the highest authority.

Rarely was it realized to what extent information is a total enterprise resource. The majority of systems came "online" more by accident of circumstance than by overall information resource planning. Even the development of DP centers themselves followed haphazard (albeit not unpredictable) courses.

Although system design, then as now, was "not so much a science as a trade or a craft," (Yourdon, 1972, p. 72) basically the designs could be categorized into two fundamental orientations (Lyon, 1971): process-oriented systems and file-oriented systems. Because both these orientations are still common (and are occasionally applied to Database design) a few words on the subject are in order.

> The process-oriented designer tends to initiate design by identifying and defining each of the system-run [sic] units in terms of internal processing, leading to a determination of the data required by each program. Finally after each "process" unit is defined, the designer reviews his data requirements and constructs a file.
>
> The alternative to process orientation is a design which begins by an examination of the total system in terms of information. . . . The file approach defines the information elements of a system and organizes them such that the information remains valid even though the details of the process change. . . . The file-oriented designer concentrates on the basic information element of the system. . . and will design the file such that

> it establishes and maintains the logical integrity of. . ./the basic element/, the subsidiary information elements, and finally the relationships among the elements (Lyon, 1971, pp. 7-9).

Even with the advent of feasible and relatively inexpensive Direct Access Storage Devices (DASD's - mostly removable disks) design orientations changed very little. Indexed Sequential, Direct Access, and in some cases, Inverted files began to be more widely used, but often even with these constructs, the analysts' view of "File" remained intrinsically the same. The heritage of sequential processing proved hard to shed.

With sequential files (particularly on magnetic tapes) it made good sense to pack each record with as much data as possible. This reduced the necessity of storing large amounts of data in the computer memory and reduced the necessity of searching files for "further pieces of data". The notion of a "masterfile" was often a large sequential file wherein each record contained tens (and sometimes hundreds) of data items. One rationale for this was that if a file included all potentially needed data, the file became more "flexible" i.e. would not need additional file interaction and would be relatively free from modification. Typically, various applications in a system would sort the masterfile (or some extracted subset of it) according to a particular

need and then process the sorted file.

This same type of thinking was carried over into non-sequential (and indexed sequential) files. Many not-strictly-sequential files were built around enormous record sizes for precisely the same reasons that sequential files were. There were, of course, exceptions,[1] but these were a small minority.

The pre-Database world was one of mostly sequential files and sequential file orientation. Systems (and sometimes subsystems) were viewed as being independent entities and the private domains of their owners. Payrolls and other basically sequential applications were functioning well, however the problems were myriad and multiplying. In the following section, some of these problems will be examined.

II. Some Problems with the Traditional Designs

Depending upon the individual's personal orientation, different authors have taken various views as to the types of problems which gave rise to the need for Database technology. Although many cite the same "source of the problem", each then proceeds to expound upon a different problem. This is understandable since one "source" may cause many problems. There seem to be

---

[1] These foresighted exceptions laid the foundations for Database technology.

three basic orientations: management-oriented, system-oriented, and machine-oriented. The following brief review of the problems first lists some of the significant "problem sources", then defines them (where necessary) and finally discusses the problems they cause for each sector of interest.

Parochial View of Data - (This has been amply elucidated in Section I).

Data Redundancy - The storage of one "piece of information" in multiple locations. There are really two types of data redundancy: explicit and implicit. Explicit redundancy occurs where one has multiple storage location of exactly the same piece of information. Implicit redundancy occurs where a piece of information and all the factors necessary coexist in storage.

Data-Dependent Applications - "This means that the way in which the data is organized on secondary storage and the way in which it is accessed are both dictated by the requirements of the application, and moreover that the knowledge of the data organization and access technique is built into the application logic" (Date, 1977, p. 10).

Lack of Flexibility in Current Data Structures - This manifests itself in a number of areas: difficulties in having "shared files", problems with restricting access on a record and/or field level (as opposed to on a file level), and problems with viewing a given file as being able to accommodate various organizational structures (although to a certain extent Indexed Sequential and Inverted files both provide the designer with some powerful options).

The outstanding problem for those of a management

bent is the fragmentation and lack of centralized control of a very valuable resource - information. As C.J. Date (1977, p. 6) points out in answering the question, "Why Database?":

> One answer is that it provides the enterprise with centralized control of its operational data - which . . . is its most valuable asset. This is in sharp contrast to the situation which prevails in most enterprises today, where typically each application has its own private files - quite often its own private tapes and disk packs, too - so that the operational data is widely dispersed, and there is little or no attempt to control it in a systematic way.

Other authors also stress this issue:

> An enterprise needs to manage its data resource. . . . Recognition that data are valuable and that they are not the property of a single function is to acknowledge the enterprise as an integrated whole and not a collection of independent units (Lyon, 1976, pp. 2-3).

Thus with the growing concern for Management Information Systems (MIS), and the growing awareness of information as an important resource, the inadequacies of pre-Database systems became quite apparent.

The machine-oriented analysts tend to emphasize some of the problems caused by data redundancy, particularly inflated file size. Overly large files are both a waste of precious storage space and result in slower response times. In today's systems "files are large; thus, redundant data must be factored out to reduce the

time required to access desired information" (Katzen, 1975, p. 165). With the quantity of information being stored increasing at an exponential rate [and the prospects "that the exponential growth could continue for a decade or two" (Martin, 1977, p. 4)] the problems of efficient data storage become increasingly more critical.

System-oriented authors have pointed out many problems with the traditional modus operandi. Here are only a few of the principle ones.

The High Cost of Data Redundancy - Obviously if multiple copies of a piece of information exist, this requires the system to perform multiple updating operations. This may not be significant when dealing with small inactive files, but the cost can be quite appreciable with large and/or highly active files.

The Increased Probability of Transcription and Updating Errors Due to Data Redundancy - There is a corollary to Murphy's Law: "Transcription errors increase as the square of the number of times a piece of information is manually transcribed." Note the lemma which states that: "the more places a given piece of information resides, the greater is the probability that one or more of its occurrences will be updated incorrectly (due to transcription errors and/or simply being overlooked)." Taken together it can be concluded that: the more the occurrences of a given piece of information, the greater the probability that one (or more) of those occurrences will contain an error (Q.E.D.).

Data Inconsistency Due to Redundancy - Even if one does manage to keep one's files free from real errors (transcription and the like), redundancy can cause the appearance of apparent errors, a condition which can be equally

embarrassing for a DP center. It is rare to find a traditional (non-integrated, non-Database) situation in which every occurrence of a redundant data item can be updated at the same time. "In a large data processing operation without a database there are so many redundant data that it is virtually impossible to keep them all at the same level of update. Too often the users or general management notice the apparent inconsistencies that this causes and distrust the computerized information. Inability to keep redundant data in the same state of update is a common cause of the anti-computer stories that managers too often tell" (Martin, 1977, p. 23).

Growth Resistant Systems Due to Data Dependent Applications - In an environment where the data organizational structure is coded as an integral part of the application programs (as in the "File Section" of a COBOL program) any change in a file, be it ever so minor, necessitates modification and recompilation of <u>every single program</u> which accesses that file. In large systems where many programs may use a single file even a minimal change in a file definition can result in tedious and expensive program modifications.

Difficulty in Sharing Stored Data - The basic alternative to private files is shared files. Traditional file organizations, however, lack the requisite flexibility for conveniently sharing files. Shared information also requires much more stringent security and privacy features than traditional file structures offer. If the golden age of integrated systems is to successfully replace the private data fiefdoms, more powerful file organizations must exist.

III. <u>Database Ideals (general goals)</u>

Having noted the drawbacks of non-Database systems and organization structures, the next step is to

examine the features of a data storage scheme which would be able to overcome the traditional difficulties and better accommodate DP needs. The convenient buzz word for such a schema (already freely used in this work) is Database. As was noted in the discussion of the traditional problems, different authors cite various features as being those which characterize a Database [e.g. CODASYL's Data Base Task Group (1971) lists twelve requisite features, C.J. Date (1977) lists seven, and James Martin (1977) lists thirty].

Such lists are never really mutually exclusive. Individual authors use different degrees of generalization and implied assumptions in enumerating their own criteria. Nevertheless, an industry-wide dispute does exist over just what constitutes a Database (and the issue is far from settled). Some of the repercussions of this dispute will be mentioned later on, but for the nonce, controversy should be avoided. A general (intuitive) idea of some desired Database features will certainly suffice for an understanding of this thesis.

To this end it is convenient to borrow a definition of Database from James Martin (1976) and a list of requisite features from the CODASYL Data Base Task Group [DBTG (1971)].

a) Definition of a Database (Martin, p. 4):

A collection of data designed to be used by different programmers is called a data base. We will define it as a collection of inter-related data stored together with controlled redundancy to serve one or more applications in an optimal fashion; the data are stored so that they are independent of programs which use the data; a common and controlled approach is used in adding new data and modifying and retrieving existing data within the data base.

b) Requisite Data Base Features (p. 6):

The Data Base Task Group's objective in developing these proposals was to make this /Database/ possible by providing features which:

- allow data to be structured in the manner most suitable to each application, regardless of the fact that some or all of that data may be used by other applications - such flexibility to be achieved without requiring data redundancy.
- allow more than one run-unit to concurrently retrieve or update the data in the database.
- provide and permit the use of a variety of search strategies against an entire database or portions of a database.
- provide protection of the database against unauthorized access of data and from untoward interaction of programs.
- provide for centralized capability to control the physical placement of data.
- provide device independence for programs.
- allow the declaration of a variety of data structures ranging from those in which no connection exists between data-items to network structures.

- allow the user to interact with the data while being relieved of the mechanics of maintaining the structural associations which have been declared.

- allow programs to be as independent of the data as current techniques will permit.

- provide for separate descriptions of the data in the database and of the data known to a program.

- provide for a description of the database which is not restricted to any particular processing language.

- provide an architecture which permits the description of the database, and the database itself, to be interfaced by multiple processing languages.

These features, then, provide both generality and flexibility and allow the building and manipulation of data structures as complex as necessary for a given application.

IV. The Current Status of "Database" Today

Three statements can characterize the position of Database technology in today's Data Processing (DP) world:

1. Almost everybody wants a "Database".

2. There is no general agreement on a precise definition of a Database.

3. There is almost general agreement that the Database packages which <u>are</u> on the market do not meet all the criteria of what a Database should be.

If it were not costing so much time and money, the situation could be comical. But time and money are being spent in enormous quantities. Every vendor of

self-esteem feels it incumbent upon himself to market at least one "Data Base Management System" (DBMS) no matter how limited. Database books, journals and other sundry publications proliferate with a prodigious frequency. DP managers, who should by now be used to the magic of such buzz words, point with pride at their own operational "DBMSs", even when the heart of such a system is often nothing more than an Indexed Sequential file with a fancy name.

What is it that causes such a phenomenon? From whence comes the need to "jump on the band wagon" of an idea that is admittedly still vague and whose future direction is uncertain? Perhaps one reason is that many view DBMS as being the cure for all the present ills and the prevention against future ones.

It is this faith in the ability of Database that is in part responsible for much of the current controversy. Vendors, all wanting to be the first with the "magical solution", all in the heat of a fiercely competitive market, invested great sums of money into developing Database technology. Many have succeeded in producing marketable packages, but like Pandora's box, there are still plagues mixed in with the blessings. Very little cooperation and coordination took place in the early development stages. The attempts

at standardization of features came too late. Vendors who have tread disparate paths do not want to retrace their steps to follow the "standard" road; they have gone too far and too much has already been invested.

This leaves the users and potential users in a difficult predicament. Most Database packages which are produced by the major hardware vendors are extremely hardware dependent. A user of IBM's IMS, for example, is totally locked into IBM (and IBM compatible) hardware. Even with some of the more machine independent packages (produced for the most part by software houses) a user gets locked into a particular piece of software. To switch to another DBMS package would require a horrendous conversion effort.

Independent groups such as CODASYL have made valiant efforts to bring about standards and standard compatible features for all Database packages, but for the present the battle lines are still drawn. Eventually the standards must come and from the current trends it appears as if CODASYL's standards will carry the day ["Nearly all of the database packages developed in the last four years used the CODASYL specification as their guide"(Towner, 1978), but with a giant like IBM leading the opposition, CODASYL's victory is far from assured].

Whatever the eventual outcome, the standards

which do emerge, forged in the crucible of controversy, will be a welcome boon to the entire DP industry. We of the present, however, must live with the uncertainties of the future and make due with the fuzzy, sometimes partisan definitions of the here and now.

It turns out, that even after one has agreed to a certain set of standard features, finding a package which truly supports these features is another matter. When the first Database packages were developed much of the requisite technology (both hardware and software) was not available. These early packages tended to strain machine capabilities to their limits and out of necessity these original packages were extremely "machine-efficiency-oriented".

This has, in the long run, proved counter-productive to the general Database goals, saddling the fledgling technology with a set of misdirected early traditions. As some commentators have noted:

> There is, however, an irony present in that the development styles of the original DBMS have tended to weaken the goals of program, and particularly programmer, independence.
>
> Although the original data base packages were conceived in the glow of program independence, they were delivered by practitioners of machine/core optimization. The good intention of program independence through DBMS was itself compromised by design and implementation goals that emphasized efficient use of machines rather than people, despite the changing machine/people

> relationships which brought them into being (Wood and Chamberlain, 1978).

In current practice there is an unfortunate perpetuation of the early machine-efficiency-orientation. We shall see this orientation again reflected in some of the Database design techniques discussed in Chapter Two.

Another factor which impedes the potential users in the search for a standard package is the perfidy of unscrupulous vendors. Often a vendor will advertise "CODASYL DBTG Standard" DBMS packages only to have short changed the user on some of the more important festures. In practice this credibility gap only serves to muddy waters even further and place one more obstacle on the path to standardization.

V. Database Schema Design

Despite the lack of unanimity among the Database package vendors (as well as among the academicians) there are certain aspects of Database design which can be relatively independent of the particular Database package to be used. One such aspect is Database schema design, the primary topic of this thesis.

> If the function of a data base were merely to store data, its organization would be simple. Most of the complexities arise from the fact that it must also show the associations between various items of data that are stored (Martin, 1977, p. 60).

The task of the Database schema is to detail exactly

what types of data will be stored in the Database and what interrelationships exist between those data types. It can be thought of as a mapping of the information capacity of the Database, a complete description, not of the specific information the Database contains, but of the kind of information it can contain.

In order to determine the "kind of information" a Database can contain it is necessary to know three things: What data items it contains; how the data items are grouped into records; and how the records are grouped into sets. These three tasks are the essence of database schema design.

Once a schema is mapped out, the designer can easily translate almost any given "logical design" into the specific form most efficient for a given Database package. Indeed, if the designer works at the schema level (and does not begin to design at the package level to fit a certain package) the problem of package independence would be greatly reduced. Instead of reflecting the framework of a specific package a schema should reflect the innate nature of the data and the information contained therein.

APPENDIX B

FURTHER DISCUSSION OF CURRENT DESIGN METHODOLOGIES

I. The Service Analysis Tasks

In Service Analysis (SA) the problem of Database design is divided among thirteen tasks. Briefly these tasks are:

1. determine who will be the clients of the system
2. determine what services the system needs to perform for the clients
3. identify the "data objects" required to support the services
4. determine the frequency of access (usage) of each of the data objects
5. describe the information available
6. describe the major functional units involved in the system
7. describe the client groups (sub-system identification)
8. collate the above information into a Service Analysis Book
9. prepare the Data Dictionary from the information in the Service Analysis Book
10. calculate the degree of redundancy in the

system (based upon the information in the Data Dictionary)

11. prepare the "First Cut Design", a grouping of data objects into "pseudosegments" (candidates for records) on the basis of access frequency
12. prepare the "Second Cut Design", a splitting of the pseudosegments into segments (records) based upon considerations of size, number of occurrences and generic relationships (logical connections)
13. prepare the first design, the definition of the Database logical structure.

It is fairly obvious from this list where the three design functions (identifying data items, grouping into records and grouping into sets) take place. Tasks one through nine identify the data items. Tasks ten through twelve group the items into records; while both twelve and thirteen group records into sets.

II. Infological Design

The infological design deals in four (4) fundamental concepts. These are (Sundgren, 1975, pp. 18-31):

1. Objects - "Intuitively an object is something that we are interested in, something that we want to gather information about.

> Objects may or may not be physical entities. Enterprises, departments, educations, professions, leisure activities and car accidents are as good 'object candidates' as are persons, buildings, areas, pets, and motor vehicles. It is when we specify a particular infological model. . . that we decide what phenomena to include as objects."

2. Properties - Properties are one of the two fundamental concepts which can be used to answer the question "What is it that we want to know about an object? . . . Intuitively we may either want to inform ourselves about the properties of an object or about the object's relations to other objects."

3. Object Relations - "Whereas properties are tied to individual objects, object relations are tied to pairs or, more generally, n tuples[1] of objects". - An object relation is a specified relationship between two or more objects (e.g. a person and a car may be related by the relationship of "owner to chattel").

4. Time - "The fourth fundamental infological concept to be introduced is time. The most convenient procedure for specifying the set of times of a particular infological model will probably be 'per constellation type' /see below/; that is, for each phenomenon to be covered by the infological model, we specify the times of potential interest."

In addition to the fundamental concepts, the infological approach requires the aid of some "derived" concepts. A complete formal discussion of these is quite beyond the scope of this thesis, but briefly some of them

---

[1]A tuple is simply a group of related objects; therefore an n tuple is a group of n related objects.

are as follows:

1. Constellations - "Objects, properties, object relations, and times form certain basic structures, called elementary constellations, or e-constellations, which together define the whole structure and contents of the object system" (Sundgren, 1974, p. 65).

   Formally stated, the definition of an e-constellation becomes: "If x is an n tuple of objects (n = 1,2,3, . . .), y is a property or an object relation, and z is a time, then the triple x,y,z is called an elementary constellation, an e constellation; and x is called the object component, y the predicate component, and z the time component of the e constellation." (Sundgren, 1975, p. 33).

   If y is a property then the e-constellation is called a property type. If y is a relation, the e-constellation is a relational type.

2. Object Group - An object group "is the set of all objects that have, have had, or will have" a given property (Sundgren, 1975, p. 34).

3. Attributes - A set of properties is an attribute if and only if there exists an object group for which the set of properties is relevant (where relevant is defined as a set of properties wherein each property is valid for at least one object in the object group for any given time slice).

Given this discussion of objective reality as a foundation, infological theory proceeds to describe the human interaction with the Database. Humans are interested in information and so must reference the Database. Thus the area of references is also analyzed. References are the basic building blocks of messages.

> References are conceptual, mental entities which human beings use when they perceive and think about

> an object system. Each reference refers to an object system entity, the target of the reference. . . . References may be combined into reference expressions. An elementary message, or e-message, is a reference expression which has an e-constellation as the object system target" (Sundgren, 1974, p. 93).

As has been stated, infological implies user oriented. Indeed the user is asked to view the Database as a magical "black box" for which he has but to define the objects, properties, attributes, messages and so forth, never to consider the computer oriented datalogical point of view. However, "for those who are to perform the computer-dependent, datalogical design of a data base, the infological 'black box view' will not be sufficient" (Sundgren, 1974, p. 93). What then must the datalogical practitioner do in order to complete the Database design and open up the magical black box?

> In order to bridge the gap between the infological and the datalogical sphere of the general data base design theory, an elementary file, or e-file, is defined as a certain "normal" representation of an e-concept. There are three basic types of e-files; object e-files, property e-files, and relational e-files. Which type we choose for a particular e-concept, ec, is dependent upon such infological parameters as the respective frequencies and response time requirements for /inquiries/ . . . .
>
> Thus the initial datalogical design step according to the extended infological theory consists in a transformation of the set of e-concepts of the infological model into a set of object, property, and relational e-files. Then there is a set of formally well-defined file structuring operators by means of which we may transform the initial file structure into a file structure which better fits

> (a) the expected infological pattern of the transactions which will hit the operative data base, and (b) the storage and access structure of available memories.
>
> After a number of applications of the file structuring operators we will arrive at the file structure which is to be implemented. The final file structure, or file system, will contain a number of the subsystems called $\alpha\beta$-complexes, or directory/file-complexes. The internal structure of such a complex may or may not conform to some well-known file organization technique. Anyhow we will have designed our file system in a much less arbitrary and much more user-influenced way than is common today (Sundgren, 1974, pp. 93-94).

Before leaving this discussion of infological Database design, a few words must be said about the major concern of this thesis - the logical design of the Database. From the infological viewpoint a Database consists of three principle "subsystems" (or parts).

> Formally we may define the infological data base as a triple,
>
> $$DB = S,N,F,$$
>
> where
>
> S is a schema
> N is a nucleus
> F is a filter.
>
> Together S and N determine the set of M of messages that are contained in the data base, the information contents of the data base (Sundgren, 1975, p. 71).

The filter is not of current concern. It has the function of protecting "the data base and its users

against false messages and messages that are not meaningful according to the specifications and definitions embedded in the data base schema" (Sundgren, 1975, p. 76). Taken together the "schema" and the "nucleus" define the logical Database design (the schema in this author's terminology). The nucleus is that portion of the Database which contains the basic data items and records. The infological schema is that which defines the requisite interrelationships (sometimes sets and sometimes records).

Once again in the words of Bo Sundgren (1975, pp. 72, 74-75):

> From an infological point of view a data base schema is identical with the specification of a particular infological model. . . . Thus a schema is a statement of a set of (references to) object types, attributes, object relations, generation rules, constellation types, internal and external definition, etc.
>
> The nucleus of a data base is a set of messages that is sufficient to generate, in combination with the schema, the information contents of the data base. If no message can be removed from the nucleus without changing the information contents of the data base, we shall say that the nucleus is infologically minimal, or non redundant. As has been said before, there may be datalogical as well as infological reasons for allowing the nucleus to be redundant.
>
> Whereas the general idea of the nucleus as a kernal or subset of messages from which the other messages of the data base are derived seems clearly conceivable even from an infological point of view, we cannot always give a strictly infological

justification for considering, or not considering, a particular message as part of the nucleus. Several distinct sets of messages may, independently of each other, fulfill the infological condition, as stated above, for being a non redundant nucleus; and any set containing one of these sets as a proper subset would be a feasible redundant nucleus. Selecting one of these redundant or non redundant nucleus candidates as the nucleus of the data base is ultimately a design decision into which data-logical efficiency considerations inevitably come.

III. Entity-Relationship Design

The E-R model is constructed in three steps (Chen, 1977b, pp. 78-79):

1. "identify entity sets of interest to the enterprise"

   "An entity is a 'thing' which can be distinctly identified. According to the needs of the enterprise, entities can be classified into different entity types such as EMPLOYEE, STOCK-HOLDER. An entity-set is a group of entities of the same type. . . .

   There are many 'things' in the real world. In addition, different enterprises may view the same thing differently. It is the responsibility of the enterprise administrator to select the entity types which are most suitable for his company."

2. identify the relationship sets of interest to the enterprise

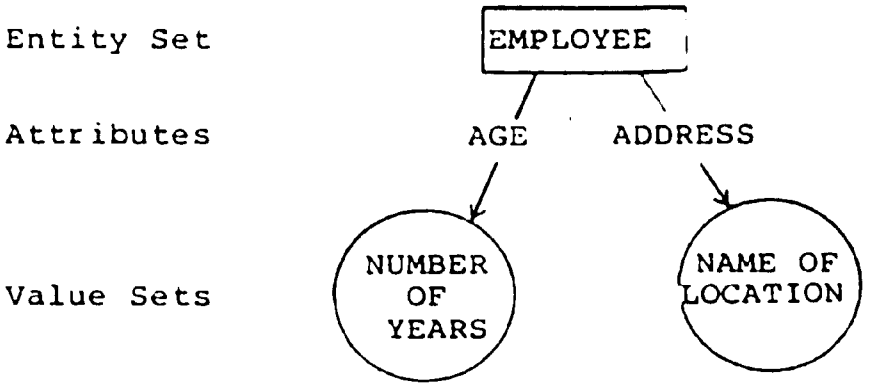
   "Entities are related to each other. Different types of relationships may exist between different types of entities. A relationship set is a set of relationships of the same type" Relationship sets may exist between two entities (e.g. employee assignments to work projects might be called the employee-project relationship set) or between multiple entities (e.g. the relationship set which could exist between the entities:

project, part-# and supplier). Relationship sets may also be of various mappings (or "ratio" types). Entities may be related on a one-to-one (1:1) basis, a one-to-many (1:M) basis, or a many-to-many (M:M) basis.

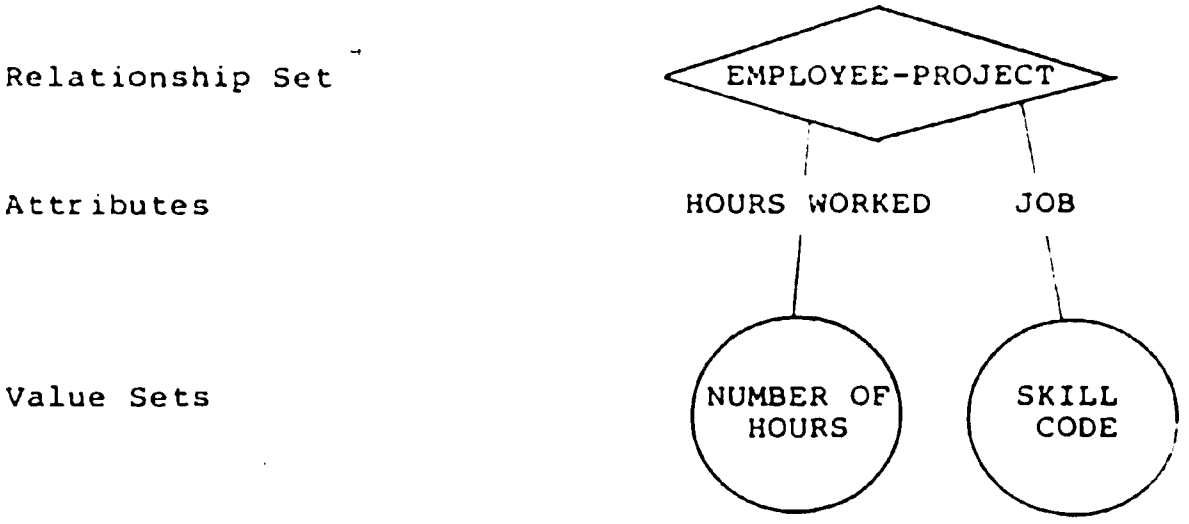
"There are many types of relationships between entities. The responsibility of the enterprise administrator is to select the relationship sets (or types) which are of interest to the enterprise."

3. "identify relevant properties of entities and relationships (i.e., define value sets and attributes)"

"Entities and relationships have properties, which can be expressed in terms of Attribute-value pairs. 'Blue', and '4' are examples of values. Values can be classified into different types such as COLOR or QUANTITY. A value set is a group of values of the same type. An attribute is a mapping from an entity set (or a relationship set) to a value set (or a group of value sets)."



It should be noted that the use of the concept of relationship set is such that relationships themselves can have attributes (and thus can be considered in some sense entities).



"It is useful to think" of the E-R model as consisting "of two conceptual domains: (1) the upper conceptual domain which consists of entity sets and relationship sets; (2) the lower conceptual domain which consists of attributes and value sets." (Chen, 1977b, p. 80) It is obvious that any given schema can never be a static completed entity. There must be mechanisms for modifications. The E-R model contains five basic modifying operations: add, delete, split, merge, and shift. Each of the first four can occur in either the upper or the lower domains. The fifth (shift) affects both domains. Add and delete are obvious functions required for the addition of new entities and the deletion of old ones from the model (with all the concomitant connection and disconnection of relationships and attributes). To split an entity is to divide it into logical sub-entities

(which now become entities). An example of a split would be the division of the entity EMPLOYEE into MALE-EMPLOYEE and FEMALE-EMPLOYEE. Relationships can also be split (e.g. the relationship EMPLOYEE-PROJECT can be split into WORKER-PROJECT and MANAGER-PROJECT). "To merge" is the reverse of "to split". The shift function handles the shifting from one conceptual domain to another. Occasionally it may be desired to view what had been viewed as a value (e.g. skill code) as an entity. This would require a shifting operation. Similarly an entity may be "downgraded" to the status of value also through the use of the shift function.

IV. Relations and Normal Forms

The basis of the relational approach is the "relation". To qualify as a relation a table must meet a number of standards. According to James Martin (1976, p. 96) the properties of a relation are:

1. Each entry in a table represents one data item; there are no repeating groups.
2. They are column-homogeneous; that is, in any column all items are of the same kind.
3. Each column (called a domain) is assigned a distinct name.
4. All rows (called tuples) are distinct; duplicate rows are not allowed.

5. Both the rows and the columns can be viewed in any sequence at any time without affecting either the information content or the semantics of the function using the table.

An example of a relation can be seen in Figure B.1. In that example the domain Employee-number is indicated to be the "Prime Key". This means that it (the prime key) is the domain which identifies the tuple; or, in other words, a tuple consists of an "object" and its attributes, the domain which identifies the object is the prime key.

Using a slightly different terminology, Gio Wiederhold (1977, p. 337) refers to the object-attribute relationship as being that between a relation's "Ruling and Dependent Parts".

> Within one relation we distinguish the set of attributes which define the object described by the tuples--the ruling part--and those which provide data regarding the object--the dependent part.
>
> In most data models it is desirable that relations have ruling parts which are unique, since we wish to have only one descriptive tuple per object. This immediately makes the tuples themselves unique. The ruling part should not contain redundant attributes; that is, it should not contain attributes which when removed would still leave a ruling part with the property of uniqueness. When an employee number is in the ruling part of a relation, the employee name is not also in the ruling part, since employee numbers are designed to be unique.

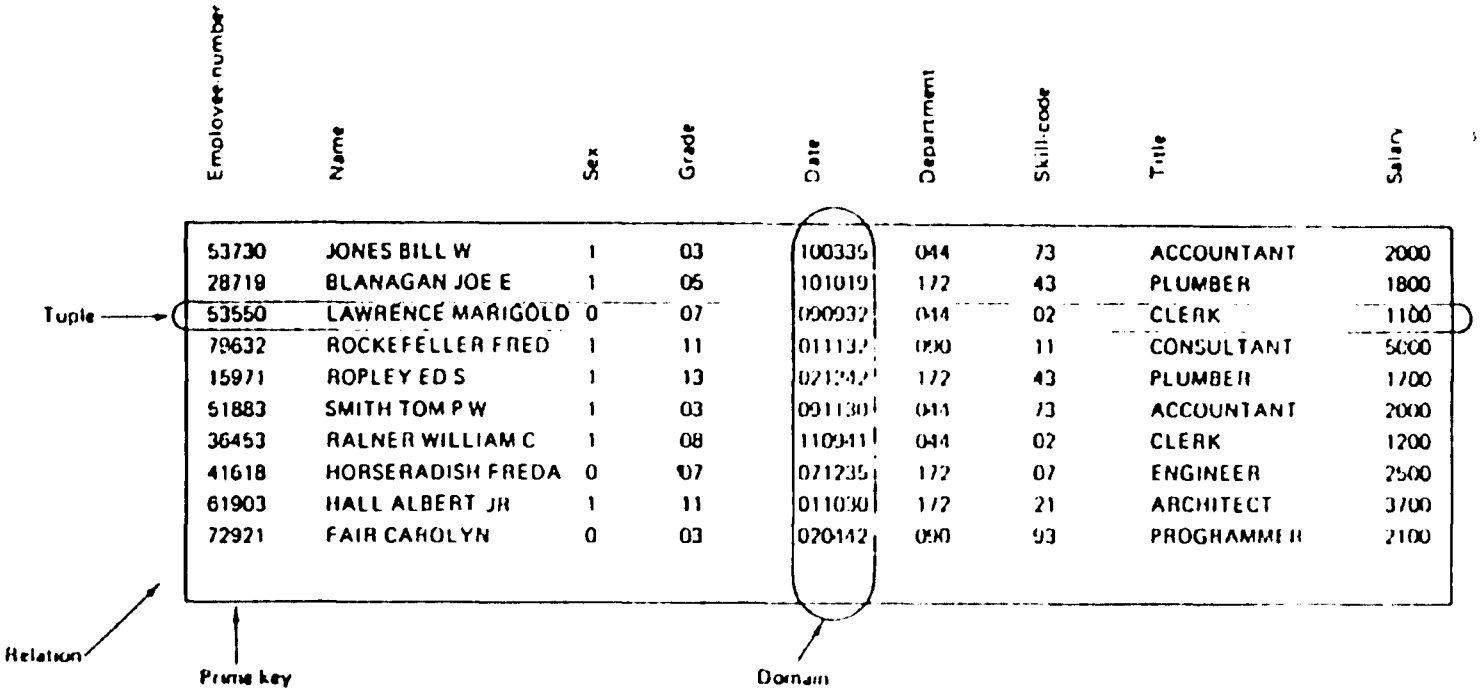


Figure B.1 "The wording used in relational data bases" (Martin, 1976, p. 97)

A relation which consists of a "normalized table" (such as Figure B.1) is considered to be in first normal form (1NF). Although simple and straightforward, a 1NF relation cannot often be readily worked with. Behind its apparently innocent exterior lurk problems which will return to haunt a designer if they are not immediately purged. These problems are not always obvious. James Martin and Gio Wiederhold each mention two such problems: relations which contain non-fully functionally dependent attributes, and relations which contain transitive dependencies. C.J. Date cites a third problem (in addition to the above mentioned): the problem of multivalued relations wherein the attributes are not fully functionally dependent in respect to one another.

Despite a somewhat foreboding vocabulary, the problems themselves are simple and readily understandable.

> If the value of one attribute B is always determined by the value of another attribute A, we say that B is functionally dependent on A. This is the relationship between the dependent part and the ruling part of a relation (Wiederhold, 1977, p. 338).

The initial process of normalization (the creation of the 1NF) does not always ensure that each non-prime attribute (each domain which is not a part of the prime key) is functionally dependent upon the

prime key. Consider the relation in this example adapted from James Martin (1977).

PART #	SUPPLIER #	SUPPLIER-NAME	SUPPLIER-DETAILS	PRICE

prime key

The prime key is the concatenation PART#-SUPPLIER#. PRICE is truly functionally dependent upon this concatenation since neither PART# alone nor SUPPLIER# alone is sufficient to uniquely identify a particular supplier's price for a particular item. SUPPLIER-NAME and SUPPLIER-DETAILS are not functionally dependent upon the PART#-SUPPLIER# key since SUPPLIER# along is sufficient to identify them; and for them, PART# is totally irrelevant. In order to remedy this situation and establish full functional dependency two (or more) relations are needed instead of one. The resulting relations are considered to be in second normal form (2NF).

PART#-SUPPLIER#	PRICE

prime key

SUPPLIER#	SUPPLIER-NAME	SUPPLIER-DETAILS

prime key

The second problem listed was that of transitive dependencies.

> Suppose that A, B, and C are three attributes of relation R. If C is functionally dependent on B and B is functionally dependent on A, then C is functionally dependent on A. If the inverse mapping is nonsimple, (i.e., if A is not functionally dependent on B or B is not functionally dependent on C), then C is said to be transitively dependent on A (Martin, 1977, p. 238).

Here again an example adapted from Martin is useful in clarifying the point. Note the relation:

EMPLOYEE#	EMPLOYEE-NAME	SALARY	PROJECT#	COMPLETION-DATE

prime key

In this case, although EMPLOYEE-NAME, SALARY, and PROJECT# are all functionally dependent upon EMPLOYEE#, COMPLETION-DATE is clearly a function of PROJECT#. It is only transitively dependent on EMPLOYEE#. As before, the solution to the problem is to split the relation into two (or more) "clean" relations. The resulting relation is considered to be in third normal form (3NF).

EMPLOYEE#	EMPLOYEE-NAME	SALARY	PROJECT#

prime key

PROJECT#	COMPLETION-DATE

prime key

The final problem (that of multivalued attributes) began to make its appearance in the literature as late as 1975. C.J. Date (1977, p. 168) uses the following example to illustrate the point. Given the relation called CTX:

COURSE	TEACHER	TEXT

prime key

or in relational terminology: CTX (COURSE, TEACHER, TEXT). This relation is definitely in third normal form "(in fact it is 'all key')" (Date). Yet suppose that for a given course (e.g. physics) there could be one of two teachers (A and B) each of whom could use either one of two texts (x or y). The possible tuples for this state of affairs are as follows:

CTX

COURSE	TEACHER	TEXT
Physics	A	x
Physics	A	y
Physics	B	x
Physics	B	y

> It is apparent that the relation CTX contains some redundancy, leading as usual to problems over storage operations. For example, to add the information that the physics course uses a new text [z] . . . , it is necessary to create two new tuples, one for each of the two teachers (Date, 1977, p. 168).

The major problem here is somewhat subtle. Linking

COURSE, TEACHER and TEXT together as a concatenated key might seem reasonable at the outset. Nothing in 1NF, 2NF, or 3NF procedures forbids it. Nevertheless non-functional dependency has crept into the picture. C.J. Date (1977, pp. 168-169) explains the problem as follows:

> First of all, attribute COURSE of the CTX relation is said to "multidetermine" attribute TEACHER. Equivalently, we say that there is a "multivalued dependence" of TEACHER on COURSE. The meaning of these statements is basically that, although a given course does not have a single corresponding teacher (i.e., TEACHER is not functionally dependent on COURSE), nevertheless each course does have a well-defined set of corresponding teachers. More precisely, we may say that the set of TEACHER values matching a given COURSE and TEXT value pair depends only on the particular COURSE value specified--the TEXT value specified is irrelevant. (As a counterexample, consider the familiar relation SP(S#,P#,QTY). Here QTY is not "multidependent" on S#, because the set of QTY values--actually a single value--matching a given S# and P# value pair certainly does not depend on the S# value alone. We note that attribute TEXT of CTX is also multidependent on COURSE; multivalued dependencies generally appear together in pairs in this way.
>
> Functional dependence, . . . is a special case of multivalued dependence. The problem with 3NF relations such as CTX is that they involve multivalued dependencies that are not also functional dependencies.

As usual, the problem can be solved by dividing the relation CTX into two relations: CT(COURSE, TEACHER) and CX(COURSE,TEXT). This results in relations of the fourth normal form (4NF). 4NF is today considered the form that a relational designer can safely work with.

To quickly recapitulate, given a normalized relation (1NF), here is what one must do before one can begin to work with it (note that in relational parlance a "projection" is a subset of the domains of a given relation).

a) Take projections of the original 1NF relation to eliminate any nonfull functional dependencies. This will produce a collection of 2NF relations.

b) Take projections of these 2NF relations to eliminate any transitive dependencies. This will produce a collection of 3NF relations.

c) Take projections of these 3NF relations to eliminate any multivalued dependencies that are not also functional dependencies. This will produce a collection of 4NF relations. (In practice, it is usually easiest to eliminate such dependencies before applying the other two normalization steps.) (Date, 1977, p. 169-170)

Given a complete set of 4NF relations, the logical structure (the information capacity) of a Database is fully described. Various user views can be accommodated through the use of relational algebra and/or relational calculus (with such operators as select, project, and join).

APPENDIX C

EXAMPLES OF THE METHODOLOGY FOR CONCEPTUAL SCHEMA DESIGN

I. Introduction

The following pages contain examples of the methodology for conceptual schema design. These are simple paradigms, chosen to illustrate various aspects of the design process rather than to display actual full scale information systems.

II. An Order-entry System

This example is based upon a problem presented by C.J. Date (1977, p. 171, 493). His word description scenario of the problem is as follows:

> A database used in an order-entry system is to contain information about customers, items, and orders. The following information is to be included.
>
> - For each customer
>   Customer number (unique)
>   Valid "ship-to" addresses (several per customer)
>   Balance
>   Credit limit
>   Discount
>
> - For each order
>   Heading information: customer number, "ship-to" address, date of order
>   Detail lines (several per order), each giving item number, quantity ordered

- For each item
  Item number (unique)
  Manufacturing plants
  Quantify on hand at each plant
  Stock danger level for each plant
  Item description

Semantic Assumptions

- No two customers have the same ship-to address.
- Each order is identified by a unique order number.
- Each detail line within an order is identified by a line number, unique within the order.

A. Phase I - Identification of the Information to be Modeled

In the above word description three distinct user views are evident. These are:

1. the customer view
2. the order view
3. the item view.

One method to graphically represent the information to be modeled is to convert the word description into traditional file record formats. Depending upon the sophistication of the environment, these can take on various forms. Figure C.1 illustrates one traditional solution to the problem. This solution involves the use of three files:

1. a customer file - with a variable length record whose length depends upon the number of addresses a customer may have

2. an order file - consisting of two record types, a header record and multiple line records

3. an item file - another variable length record. Its length depends upon the number of plants which manufacture a given item.

CUSTOMER VIEW

CUSTOMER#	BALANCE	CREDIT-LIMIT	DISCOUNT	NO-OF-ADDRESS	ADDRESS

ORDER VIEW

ORDER-ID	HEADER-ID	CUSTOMER#	ADDRESS	ORDER-DATE

ORDER-ID	LINE#	ITEM#	QTY-ON-ORDER

ITEM VIEW

ITEM#	DESCRIPTION	NO-OF-PLANTS	PLANT	QTY-ON-HAND	DANGER-LEVEL

Figure C.1 Traditional File View

Unfortunately although this traditional representation accurately illustrates one possible physical file organization, it leaves many questions unanswered. It does not (nor can it) represent the information needs (the information use) which the stored data must provide. Will, for example, it be necessary to find all customers

with a certain credit limit (i.e. view customer as an attribute of credit limit)? Would it ever be necessary to know, in total, how many of a certain item are still on order, or on which orders a certain item appears (i.e. view order as an attribute of item number)? These are aspects of the system which remain ambiguous; ambiguities which only an intimate knowledge of the system can dispel. In actual practice an analyst would interact with the "users" of the system in order to clear up these uncertainties, but for the present example an arbitrary choice of system definitions will suffice.

For an unambiguous representation of the system's information needs, bubble charts are used. Figure C.2 illustrates the three user's views of the system in bubble chart format. Some of the significant assumptions made in forming up the system definitions are as follows:

1. that there is no need to retain a data item "HEADER-ID" since the determination of which information belongs in the header and which in the detail lines can be determined within an application program.

2. that quantity ordered is properly a function of the concatenation of item number and order.

3. that there is a significance to an order's line number and it should be retained in the Database.

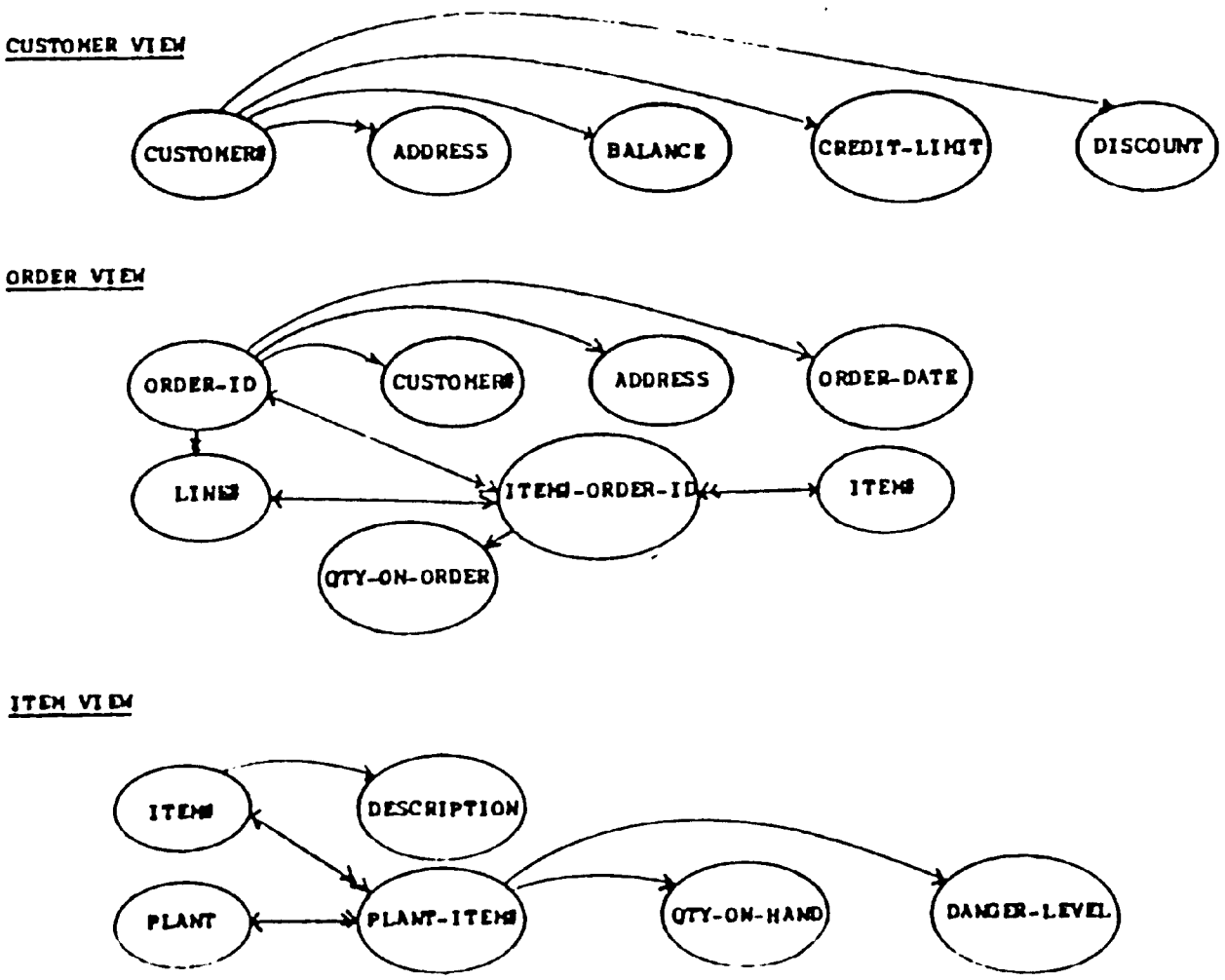


Figure C.2 The Three Views

4. that the quantity of an item on hand at a given plant, and the danger level of an item at a given plant are both functions of the concatenation of plant and item number.

B. Phase II - Construction of the Information Model

The first step in the design of the information model (the conceptual schema) is the construction of the object/attribute matrix. This can be done mechanically from each of the user's view bubble charts. Figure C.3 shows the object/attribute matrix with just the customer view represented. Figure C.4 and Figure C.5 respectively show the matrix after the incorporation of the order view and the item view (the final matrix).

Figure C.6 illustrates the conceptual schema derived from the object/attribute matrix of Figure C.5. Once again the process of constructing a conceptual schema from the object/attribute matrix is a purely mechanical one.

There is one shared attribute (ADDRESS) in the conceptual schema and the question is, should it be allowed to remain, or should a modification be made. It is this author's opinion that ADDRESS in this instance is a true case of a shared attribute. There are no grounds for asserting that the ADDRESS referred

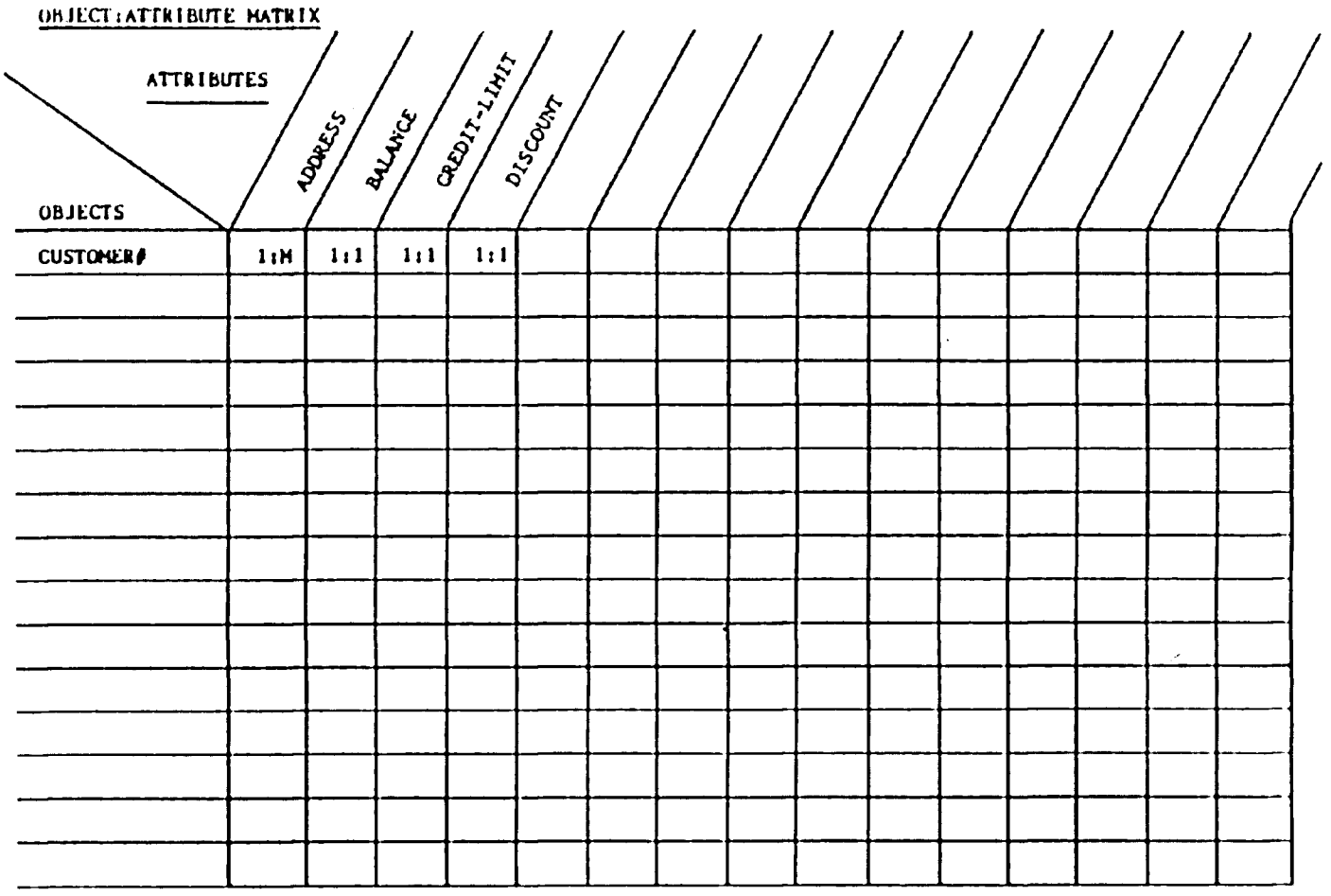


Figure C.3 Object/Attribute Matrix with one user's view.

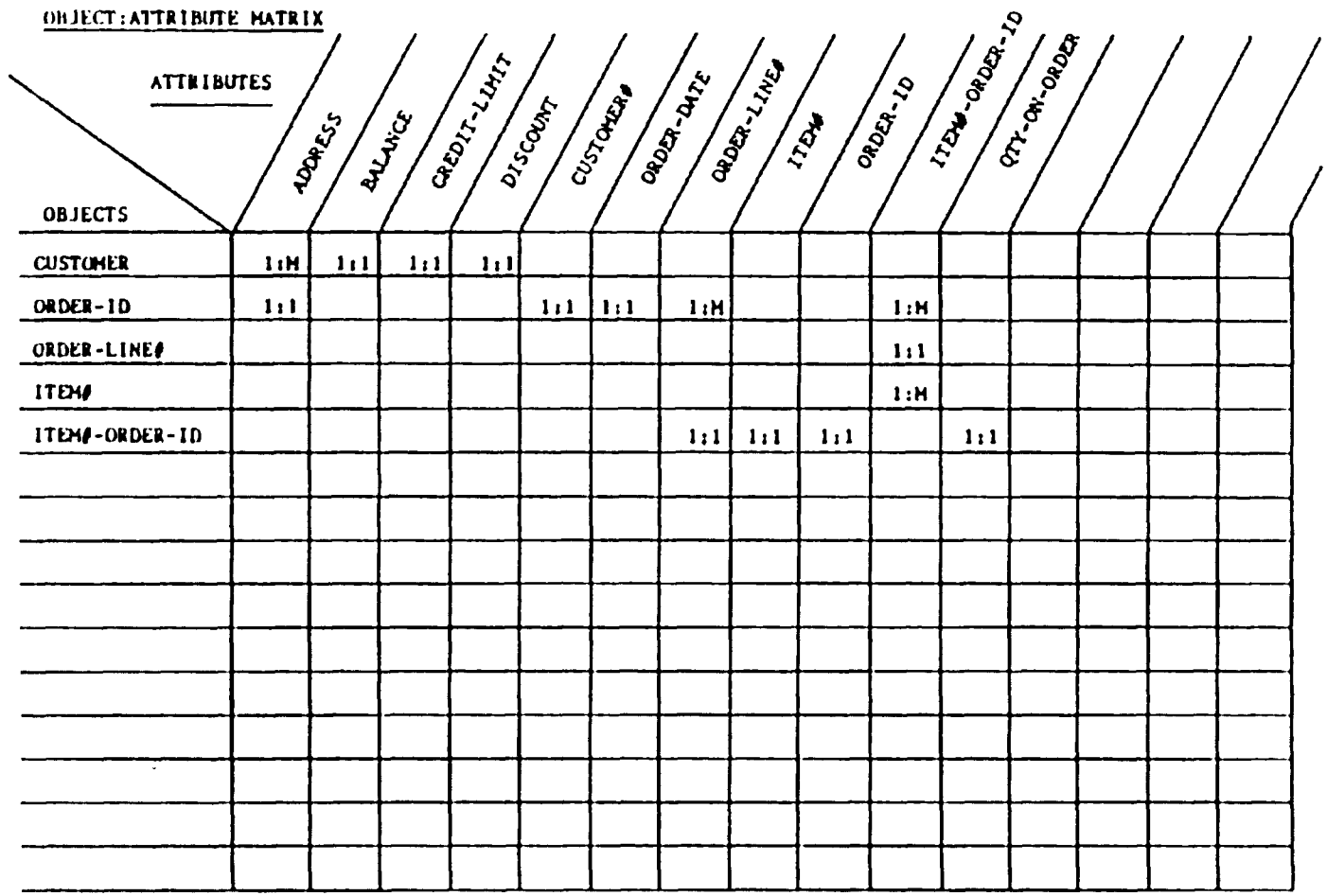


Figure C.4 Object/Attribute Matrix incorporating two users' views.

OBJECT:ATTRIBUTE MATRIX

OBJECTS \ ATTRIBUTES	ADDRESS	BALANCE	CREDIT-LIMIT	DISCOUNT	CUSTOMER#	ORDER-DATE	ORDER-LINE#	ITEM#	ORDER-ID	ITEM#-ORDER-ID	QTY-ON-ORDER	DESCRIPTION	PLANT	PLANT-ITEM#	QTY-ON-HAND	DANGER-LEVEL	
CUSTOMER#	1:M	1:1	1:1	1:1													
ORDER-ID	1:1				1:1	1:1	1:M			1:M							
ORDER-LINE#										1:1							
ITEM#										1:M		1:1		1:M			
ITEM#-ORDER-ID							1:1	1:1	1:1		1:1						
PLANT														1:M			
PLANT-ITEM#								1:1					1:1		1:1	1:1	

Figure C.5 Final Object/Attribute Matrix

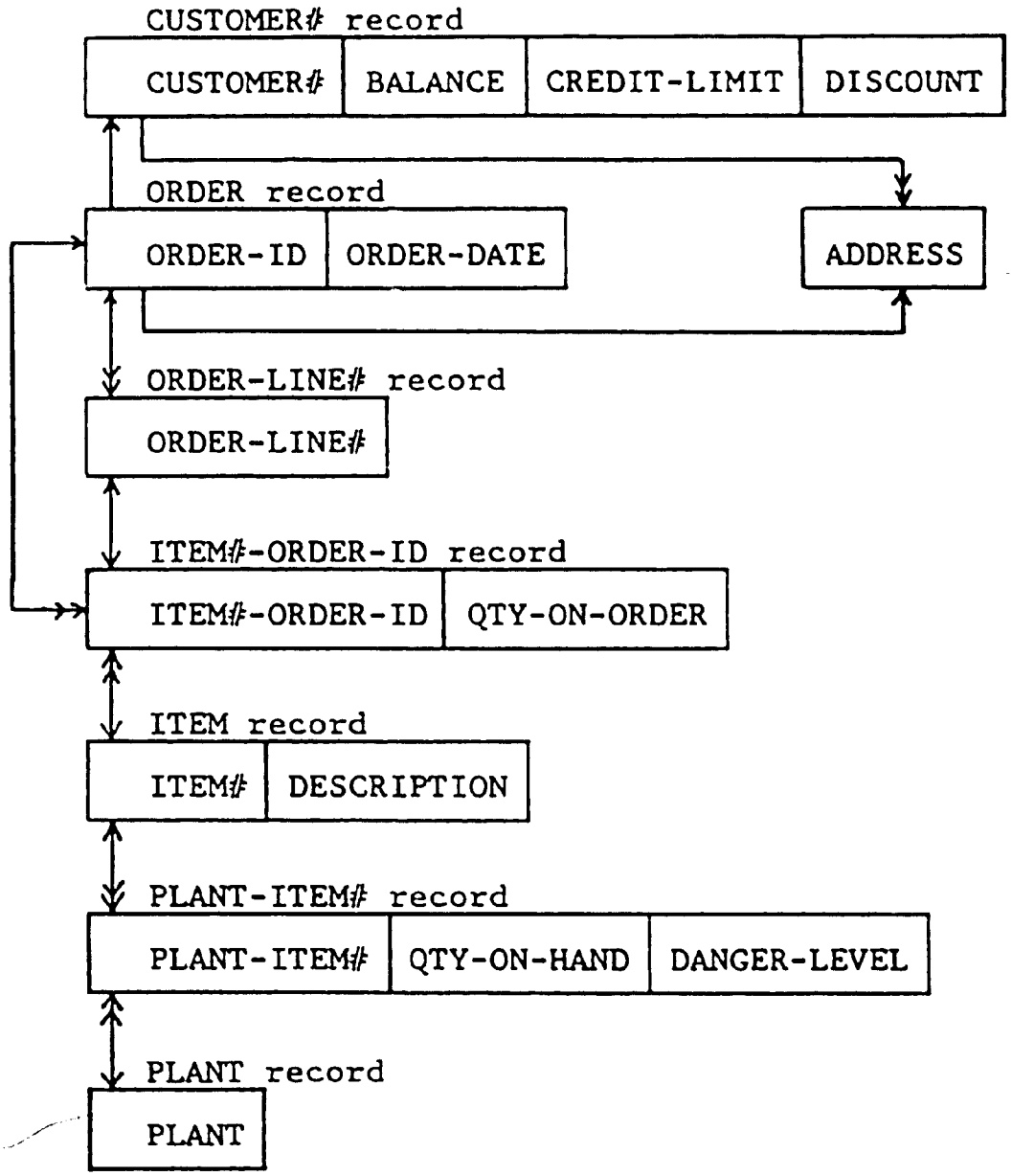


Figure C.6 Conceptual Schema

to by the CUSTOMER record is different from the ADDRESS attribute of the ORDER record; since every occurrence of an ORDER address must also be an occurrence of a CUSTOMER address. Thus, insofar as the conceptual schema is concerned, ADDRESS is a shared attribute and should remain as such (although when designing the working schema, taking physical constraints into consideration, it may be desirable to introduce some redundancy and append the ADDRESS to both the ORDER and the CUSTOMER record).

Before concluding that the schema shown in Figure C.6 is the final conceptual schema, the reader should check the original user's views against the schema to determine whether the schema can really accommodate them and whether to do so requires needless complications. In the current example it will be found that the schema easily accommodates the user's views with no complications.

III. A Cargo Fleet System

The following example involves a system of cargo ships, ports of call, shipping containers, shipping agents and all the other romantic paraphernalia of life on the high seas. The scenario is borrowed from James Martin (1977, p. 277-279) and is quoted here

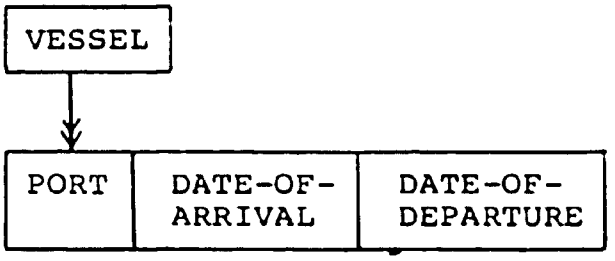
in its entirety.

First User View:

Information is stored about each ship, including the volume of its cargo storage capacity. The key is VESSEL.

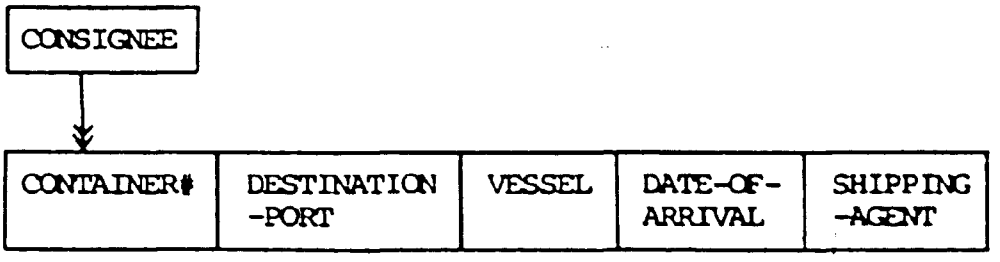
Second User View:

A ship stops at many ports and it is necessary to print out its itinerary:



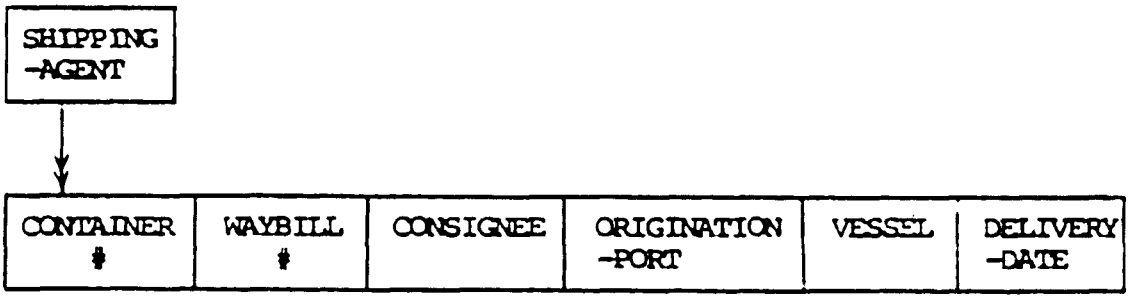
Third User View:

Persons who ship goods are referred to as consignees. Their goods must be crated or stored in shipping containers. These are given a container identification number. A list can be obtained, when requested, of what containers have been sent by a consignee:



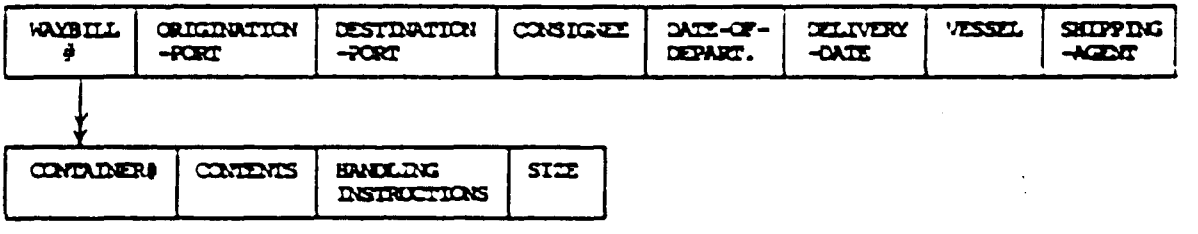
Fourth User View:

The shipments are all handled by shipping agents. A shipping-agent report must be generated, listing all the containers that a given agent is handling and giving their waybill numbers:



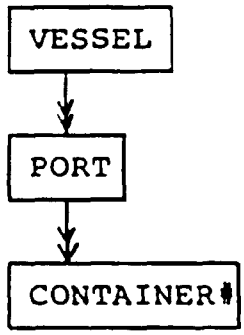
Fifth User View:

The fifth user view is the waybill. A waybill relates to a shipment of goods between two ports on a specified vessel. The shipment may consist of one or more containers:



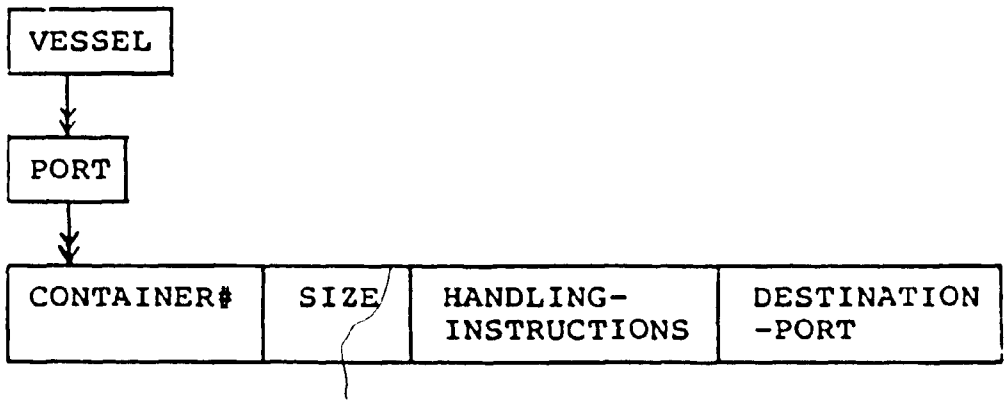
Sixth User View:

For a given vessel a list is required of what containers should be off-loaded at each port.



Seventh User View:

For a given vessel a list is required of what containers are to be loaded at each port. Details of the container size, handling instructions, and destination port are needed for loading purposes:

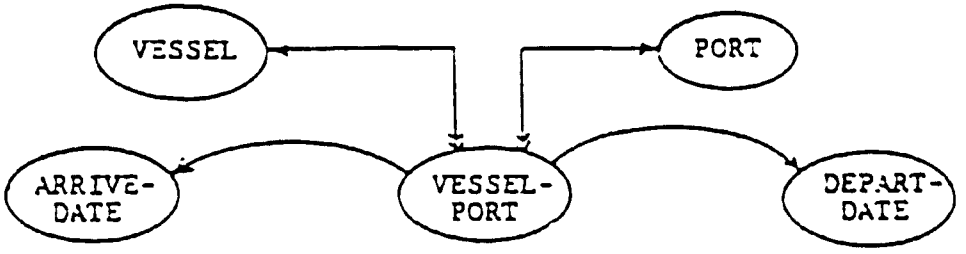


A. Phase I - Identification of the Information to be Modeled

Once again the most profitable first step in the analysis is to bubble chart the situation. Figure C.7 shows the bubble charts for each of the seven user's views. Since these views seem well defined, there is

First User View

Second User View



Third User View

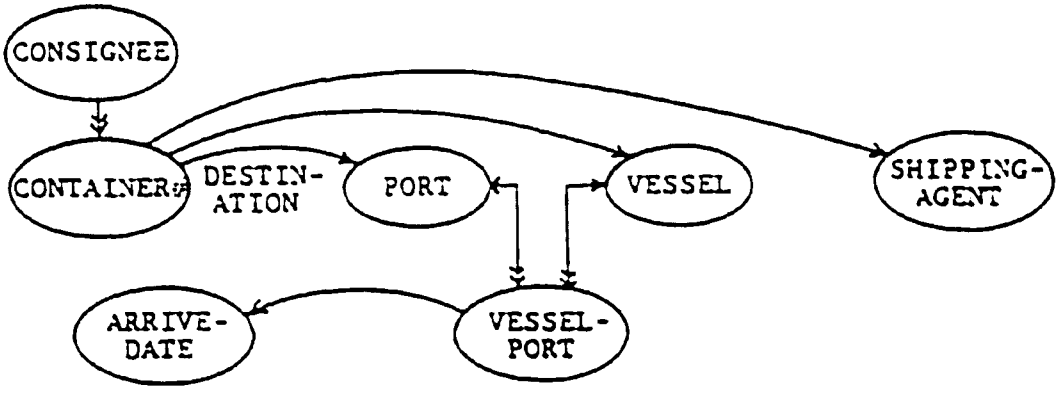
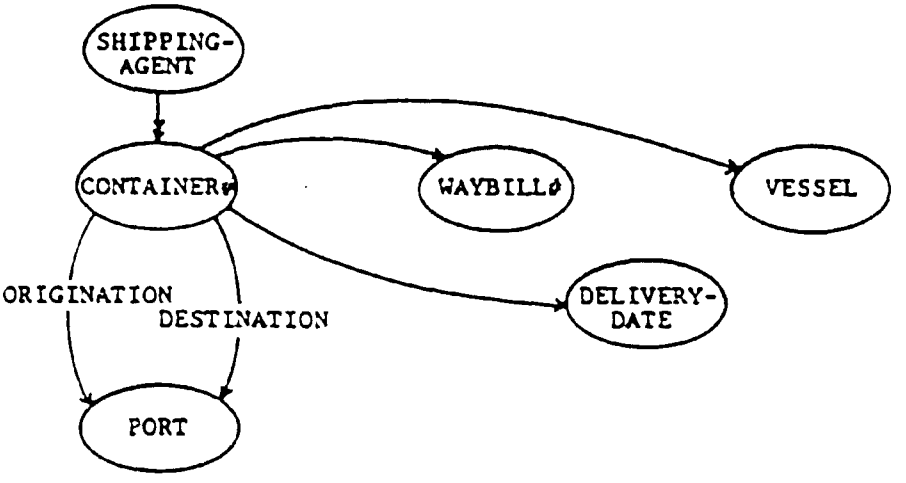


Figure C.7 Seven Users' Views

Fourth User View



Fifth User View

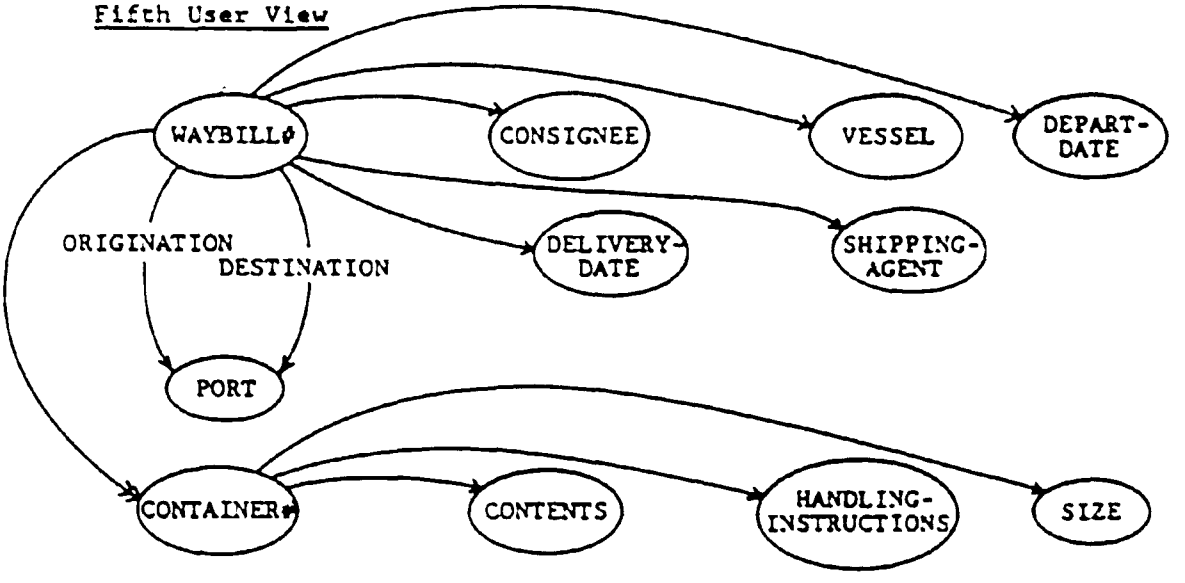
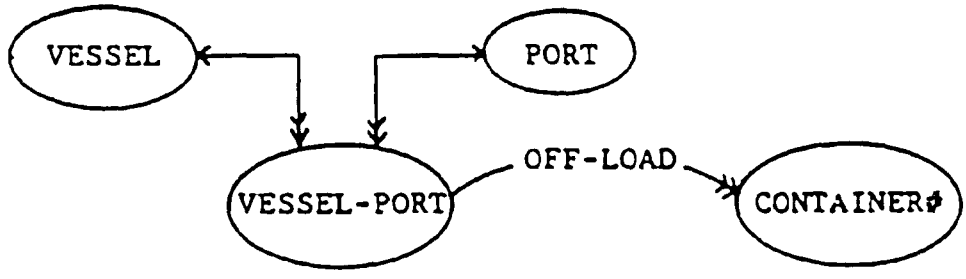


Figure C.7 continued

Sixth User View



Seventh User View

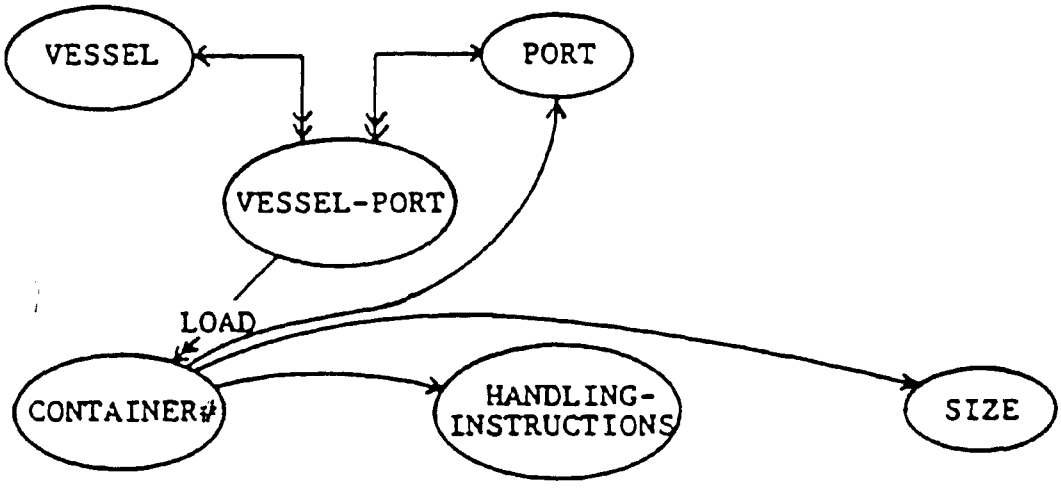


Figure C.7 continued

little else to do in this phase.

B. Phase II - Construction of the Information Model

Stage 1, the object/attribute matrix, follows directly from the bubble charts. Figures C.8, C.9, and C.10 show successive stages in the construction of this example's object/attribute matrix (C.10 is the completed matrix).

The initial schema design, (stage 2) is shown in Figure C.11. This schema, a direct mapping from the object/attribute matrix, appears exceedingly complex. Usually when a schema appears to be this complex something is wrong, and in the current example there is a great deal that is wrong.

The problem lies in the Phase I analysis (as indeed Phase II so far has done nothing but represent that analysis in a graphical format). A major source of difficulty is that those views which are concerned with the relationships of the CONTAINER#, the WAYBILL#, the CONSIGNEE, and the SHIPPING-AGENT have for the most part ignored the critical role that a WAYBILL plays in the system. The formulation of these user's views have made the common error of deriving a "view" on the basis of only a report format and not on the basis of the actual information structure. WAYBILL is a central concept to the system. It is the connection between a

OBJECT:ATTRIBUTE MATRIX

OBJECTS \ ATTRIBUTES	CARGO-VOLUME	DETAILS	VESSEL	PORT	VESSEL-PORT	ARRIVE-DATE	DEPART-DATE	CONTAINER#	SHIPPING-AGENT							
VESSEL	1:1	1:1			1:M											
VESSEL-PORT			1:1	1:1		1:1	1:1									
PORT					1:M											
CONSIGNEE								1:M								
CONTAINER#			1:1	1:1					1:1							

Figure C.8 The Object/Attribute Matrix for Users' Views #1 through #3

OBJECT:ATTRIBUTE MATRIX

OBJECTS \ ATTRIBUTES	CARGO-VOLUME	DETAILS	VESSEL	PORT	VESSEL-PORT	ARRIVE-DATE	DEPART-DATE	CONTAINER#	SHIPPING-AGENT	WAYBILL#	DELIVERY-DATE	CONSIGNEE	HANDLING-INSTRUCTIONS	CONTENTS	SIZE
VESSEL	1:1	1:1			1:M										
VESSEL-PORT			1:1	1:1		1:1	1:1								
PORT					1:M										
CONSIGNEE								1:M							
CONTAINER#			1:1	1:1	1:1				1:1	1:1	1:1		1:1	1:1	1:1
SHIPPING-AGENT								1:M							
WAYBILL#			1:1	1:1	1:1			1:M	1:1		1:1	1:1			

Figure C.9 The Object/Attribute Matrix for User's Views #1 through #5

OBJECT:ATTRIBUTE MATRIX

OBJECTS \ ATTRIBUTES	CARGO-VOLUME	DETAILS	VESSEL	PORT	VESSEL-PORT	ARRIVE-DATE	DEPART-DATE	CONTAINER#	SHIPPING-AGENT	WAYBILL#	DELIVERY-DATE	CONSIGNEE	HANDLING-INSTRUCTIONS	CONTENTS	SIZE
VESSEL	1:1	1:1			1:M										
VESSEL-PORT			1:1	1:1		1:1	1:1	M:M	M:M						
PORT					1:M										
CONSIGNEE								1:M							
CONTAINER#			1:1	1:M	1:M					1:1	1:1	1:1		1:1	1:1	1:1
SHIPPING-AGENT								1:M							
WAYBILL#			1:1	1:M	1:M		1:1	1:M	1:1		1:1	1:1			

Figure C.10 The complete Object/Attribute Matrix - all 7 Users' Views

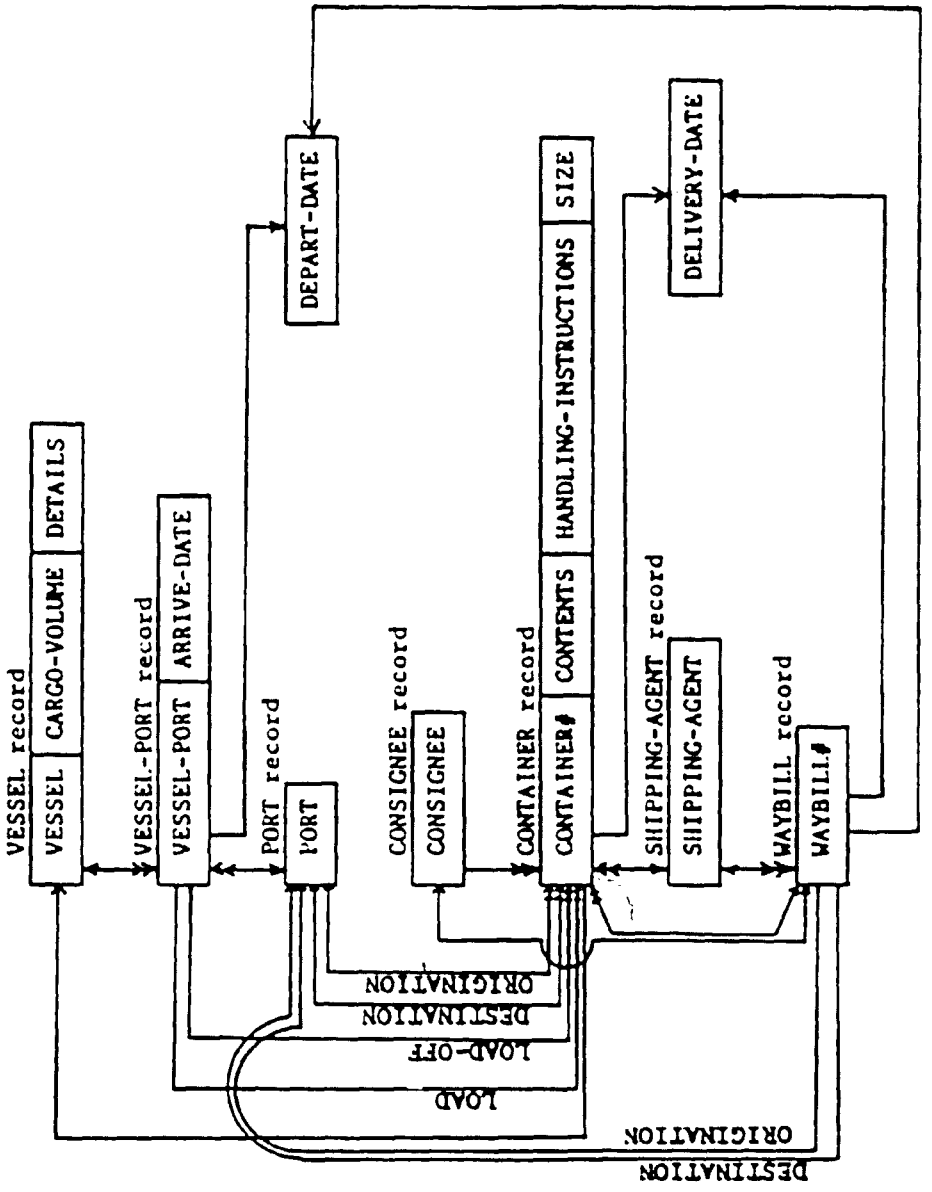


Figure C.11 Initial Schema Design

CONSIGNEE and a SHIPPING-AGENT, the connection between a CONSIGNEE and his CONTAINERs, and the connection between the CONTAINERs and their VESSEL and PORTs. A graphic representation of the importance of a WAYBILL# can be seen in the bubble chart Figure C.12.

Although there are also other minor problems with the original Phase I bubble charts (there usually are), the main concern is how to clear up the primary problem. There are two ways in which this can be done. On the one hand, the analyst can suspect that something is amiss due to the complexity of the initial schema, and perform a thorough re-evaluation of the Phase I analysis. On the other hand, it is entirely possible that the subjective "complexity clue" eludes the analyst. In such a situation it becomes the function of the Phase II stages 3 and 4 to disclose the problems and suggest solutions. For this example let it be assumed that the latter situation has occurred and that the analyst takes the schema complexity for granted and continues blythly on his way.

His first task after the formulation of the initial schema design is (stage 3 of Phase II) the examination of the use of shared attributes. In this example (Figure C.11) there are two: DEPART-DATE and DELIVERY-DATE.

DEPART-DATE is an attribute of both VESSEL-PORT

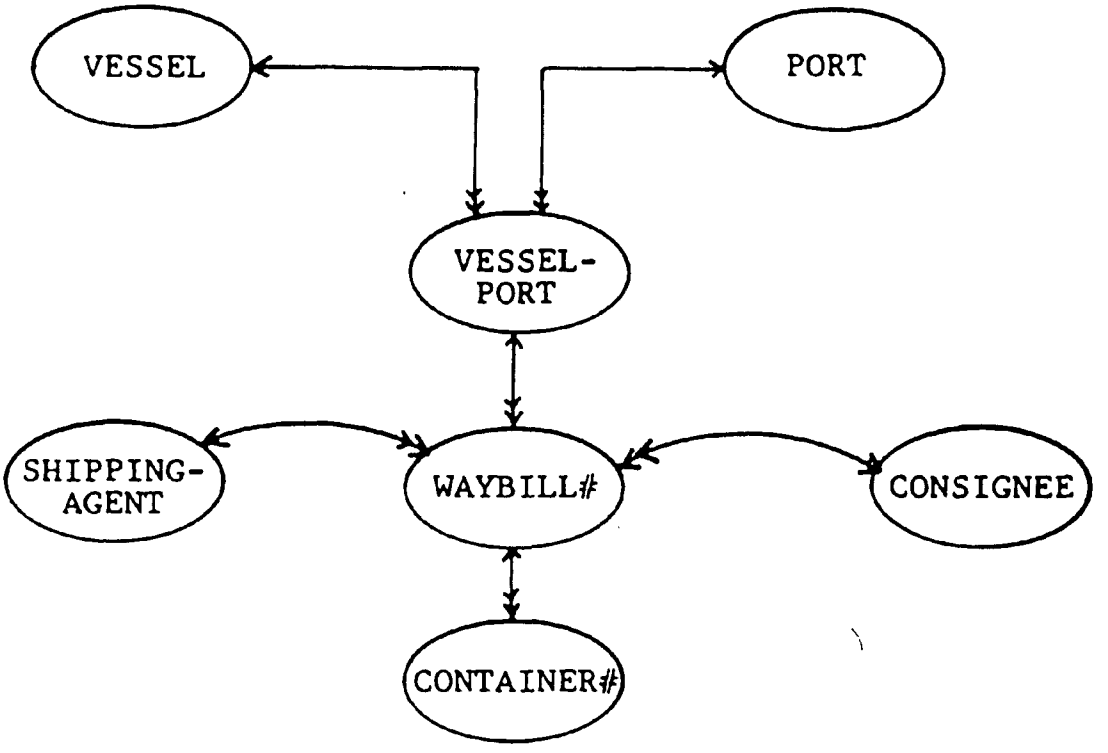


Figure C.12 The central role of WAYBILL#

and WAYBILL#. Intuitively, however, it is the VESSEL which departs from a certain PORT on a certain DEPART-DATE. A WAYBILL departs on a certain date only by virtue of its being assigned to a VESSEL. Thus in this situation, it is more accurate to retain DEPART-DATE as a unique attribute of VESSEL-PORT, and to obtain the DEPART-DATE for a specific WAYBILL# by connecting it to the appropriate VESSEL-PORT (see Figure C.13).

DELIVERY-DATE, the other shared attribute, belongs to both WAYBILL# and to CONTAINER#. Here again, an intuitive analysis should be sufficient to resolve the issue. Consider the hierarchy represented by the bubble chart in Figure C.12; containers belong to a waybill. All the parties involved in the shipping process (except perhaps the stevedores) deal in terms of shipment numbers (i.e., WAYBILL#), not in individual containers. A CONTAINER is transported by a specific VESSEL and has a specific DELIVERY-DATE, by virtue of its assignment to a specific WAYBILL#. Thus it is that DELIVERY-DATE should be a unique attribute of WAY-BILL# (and only by association can it be an attribute of CONTAINER#). This rearrangement is illustrated in Figure C.14.

Having dispensed with the shared attributes, attention can now be focused upon stage 4; consideration

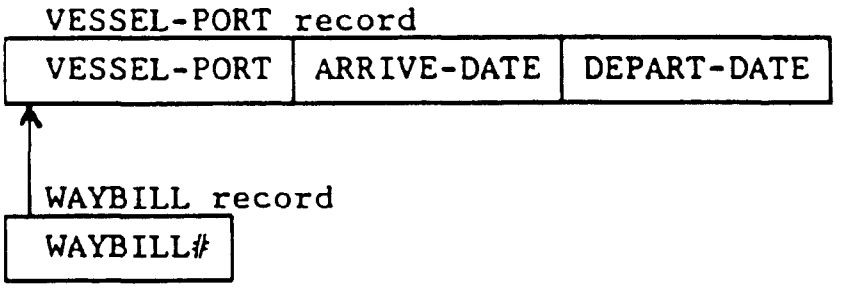


Figure C.13 Modified DEPART-DATE

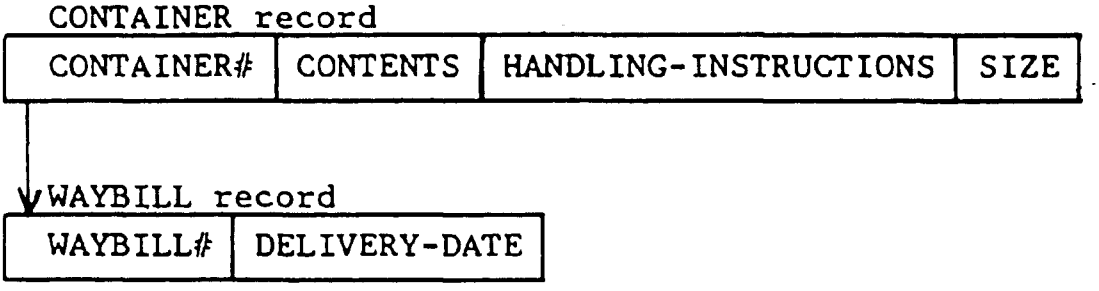


Figure C.14 Modified DELIVERY-DATE

of redundant sets (of which the example has many). Perhaps the most strikingly complicated feature of the initial schema is the mass of arrows connected to the CONTAINER# record. It is, therefore, the most logical place to begin a search for redundant sets.

Remember the stage 3 discussion of CONTAINER#: a container is never dealt with individually, but only as a part of a shipment (a WAYBILL). Thus every arrow connected to CONTAINER#, except the one between WAY-BILL# and CONTAINER#, is superfluous. The SHIPPING-AGENT deals with waybills and can obtain all his CONTAINER#s through his WAYBILL#s. Similarly the CONSIGNEE works with a WAYBILL# to trace all his con-TAINER#s. VESSELs load and unload shipments (WAYBILL#s) at various PORTs. From the crew's point of view a container is merely a part of a shipment, hence the VESSEL and PORT records should not be directly connected to CONTAINER#.

In addition to the above "intuitive" argument other sets can be seen to be redundant. There is, for example, no need for any of the sets involving VESSEL and PORT when parallel sets exist (sets from the same record) to the VESSEL-PORT concatenation. Finally, it should be noted that the pairs of sets LOAD and ORIGINATION, and LOAD-OFF and DESTINATION, are merely

complementary aspects of the same basic relationships.

Incorporating all the modifications implied by this analysis completes stage 4. Figure C.15 is the resultant schema. It is a simple and very straightforward design.

In working through this example it was pointed out above that after stages 1 and 2 of Phase II produced an overly complex schema it would have been feasible for the analyst to suspect a problem existed and to begin a re-evaluation of the Phase I results. If that re-evaluation had successfully taken place, the importance of the WAYBILL record as a key record would have been discerned. In the example, however, it was assumed that the analyst did not suspect that a re-evaluation was necessary. Nevertheless in the final schema (Figure C.15) WAYBILL# is indeed a very key record.

This illustrates an interesting aspect of the methodology, that even if the Phase I analysis is less than perfect, Phase II will often force the analyst to ask the questions which can rectify previous mistakes.

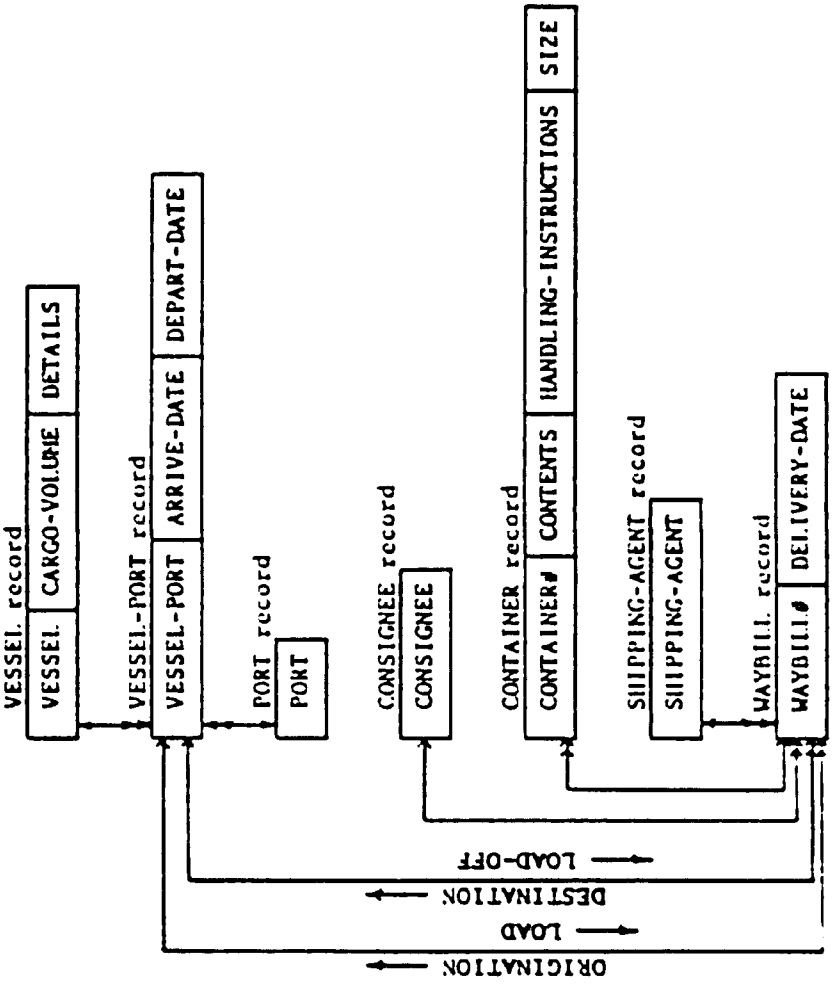


Figure C.15 The Final Schema

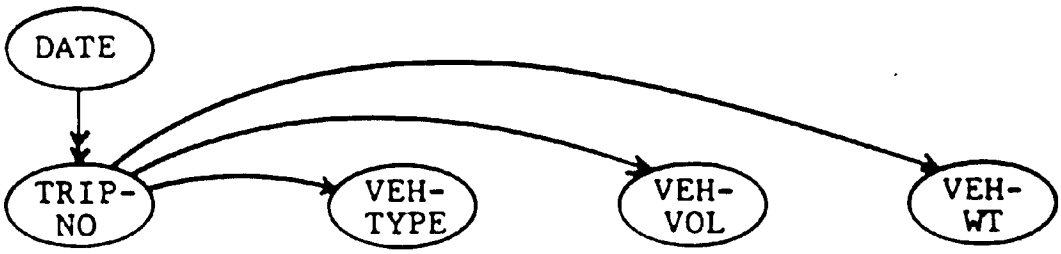
APPENDIX D

A COMPARISON WITH THE CANONICAL DESIGN APPROACH

A fair comparison between two methodologies should involve a neutral example, one that is not specifically tailored to suit the strong points of either of them. Nevertheless to stack the odds unfavorably, the example chosen for this comparison is one which the proponents of the canonical design use as a vehicle to demonstrate the effectiveness of their methodology. It is the example which Raver and Hubbard present in their article Automated logical data base design: Concepts and applications (1977).

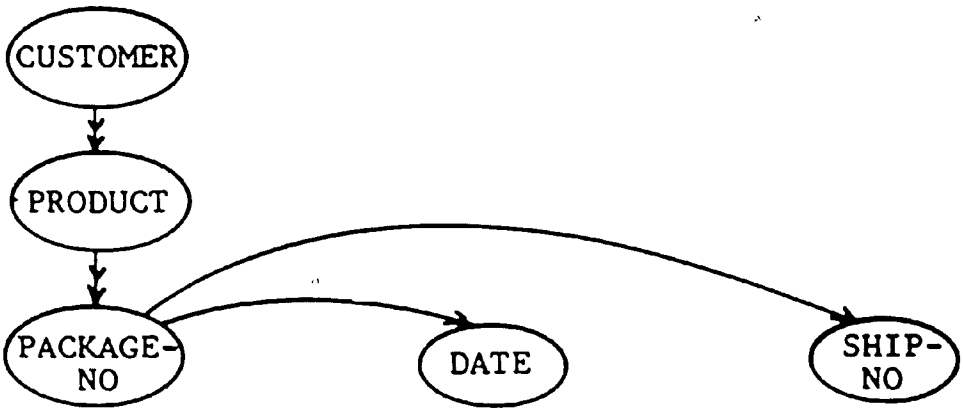
Here is the scenario as they present it:

> A data base is being designed for a trucking company that loads its trucks with products for shipment to various customers. Many trips are made each working day, and each trip is made by a certain type of vehicle. Each component of a product is given a package number. On a specific trip, all packages for a given customer are grouped and given a single shipment number.
>
> The data base is required to support five application functions that provide operating information for the company. A schematic representation of each function is depicted in each part of [Figure D.1] . . . . (For simplicity, only the output requirements of each function are considered.)
>
> Part A of [Figure D.1] . . . . shows the trip schedules (Local View 1) that list each trip by date, and for each trip, give the vehicle type,



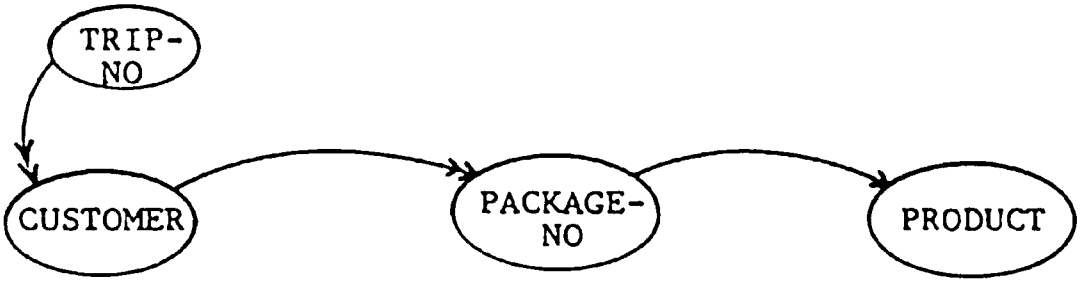
(A)

(B)

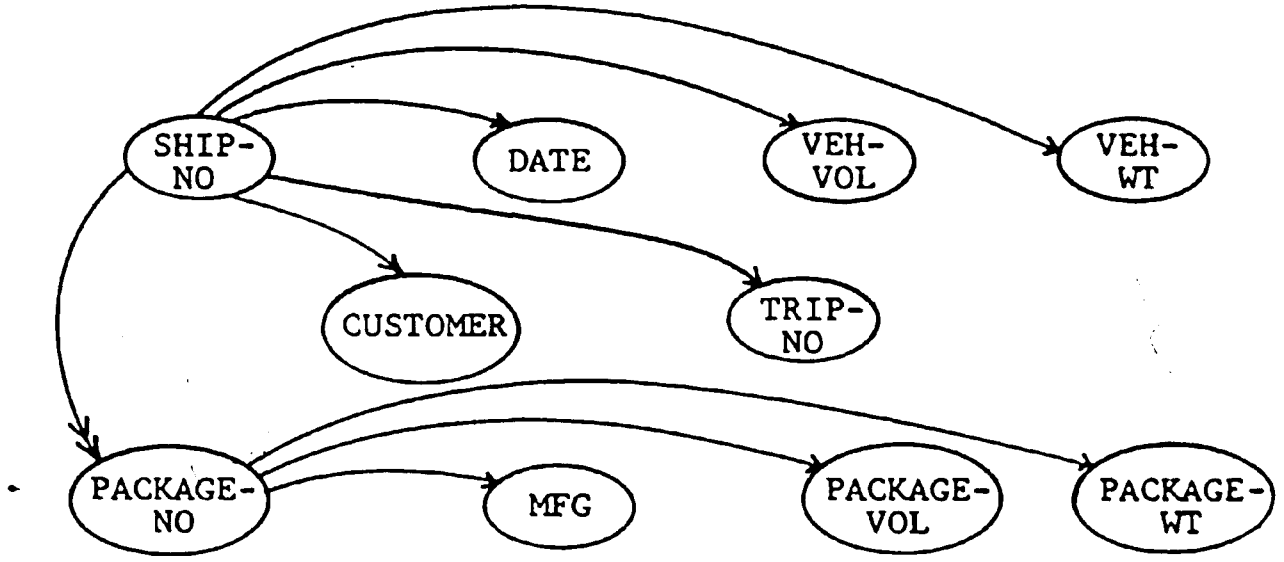


(C)

Figure D.1 Local Views



(D)



(E)

Figure D.1 continued

weight, and volume. The customer shipment query (Local View 2) shown in Part B handles customer queries about the dates of scheduled trips to a customer. Part C illustrates the customer product query (Local View 3) that handles customer queries such as, "When and what is the shipping information for given products?" The trip contents (Local View 4) lists each trip, the customers to be served, and the packages and products to be delivered as shown in Part D. The shipment history (Local View 5) in Part E provides a history of each shipment.

The schema produced by the canonical design technique is shown in Figure D.2.

Using the eidos based object/attribute methodology, the first step is to establish the users' views. Since these are already given in the problem, one can proceed directly to the construction of the object/attribute matrix. The matrix for this example is shown in Figure D.3, and the resultant schema is presented in Figure D.4.

This initial schema contains some shared attributes and redundant sets, all of which must be examined before one can conclude that the schema is in final form. The shared attributes are VEH-WT and VEH-VOL. In each case the attribute is shared by TRIP-NO and SHIP-NO. This is hardly a situation of genuinely shared attributes, as there can be many shipments involved in a single trip, and the volume and weight of the

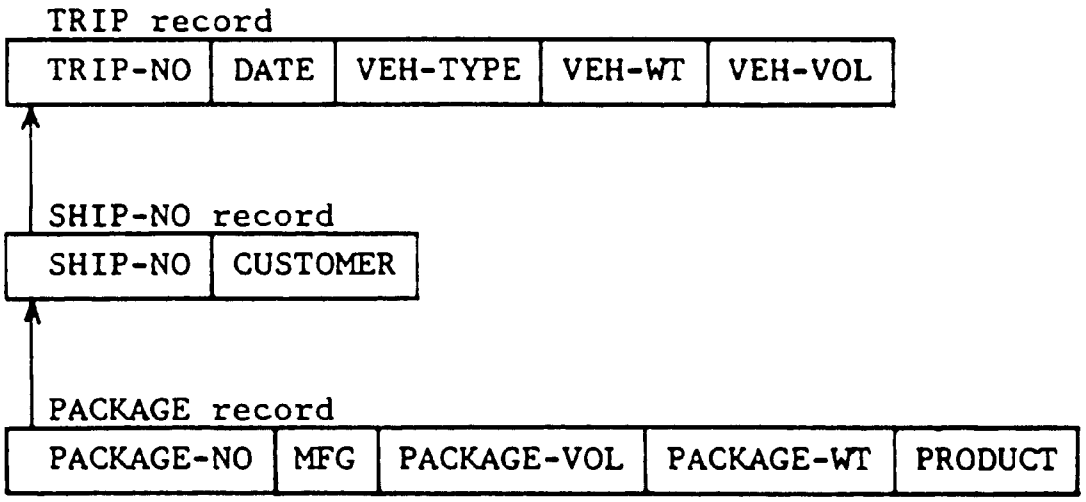


Figure D.2 Canonical designed schema

OBJECT:ATTRIBUTE MATRIX

OBJECTS \ ATTRIBUTES	TRIP-NO	VEH-TYPE	VEH-VOL	VEH-WT	DATE	PRODUCT	PACKAGE-NO	SHIP-NO	CUSTOMER	MFG	PACKAGE-VOL	PACKAGE-WT			
DATE	1:M														
TRIP-NO		1:1	1:1	1:1	1:1				1:M						
CUSTOMER	1:M					1:M	1:M								
PRODUCT							1:M								
PACKAGE-NO					1:1	1:1		1:1		1:1	1:1	1:1			
SHIP-NO	1:1		1:1	1:1	1:1		1:M		1:1						

Figure D.3 The Object/Attribute Matrix

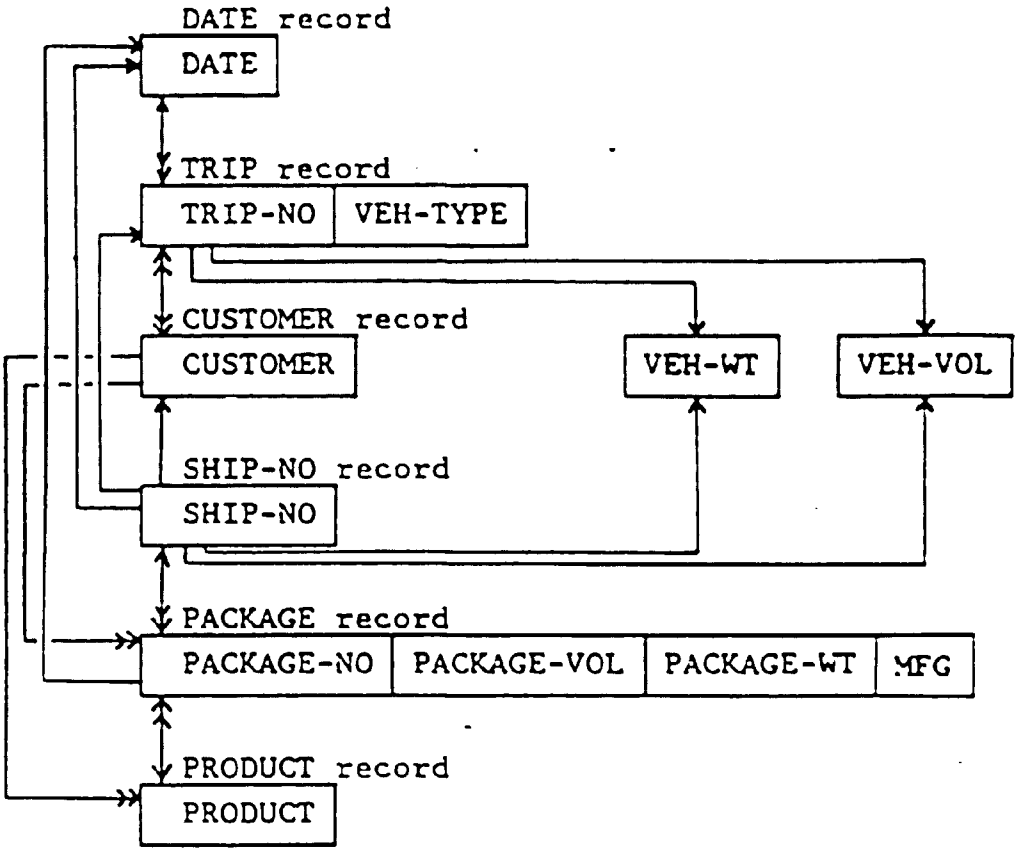


Figure D.4 Initial Schema

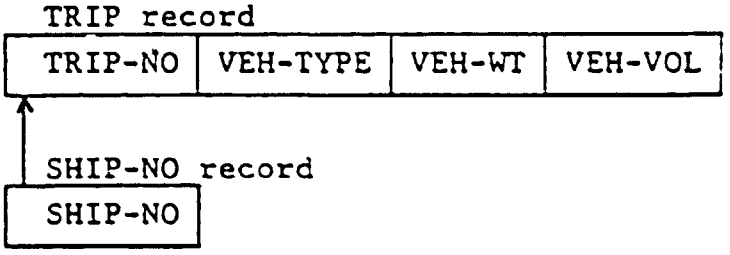


Figure D.5 Schema Modifications

transporting vehicle are independent of any one shipping number. On the other hand, the volume and weight of the vehicle used are definitely legitimate aspects of a trip description, and thus should be considered unique attributes of TRIP-NO. It is far more sensible to think of a shipment linked to a specific trip than linked to a specific vehicle volume and weight. These modifications are shown in Figure D.5.

The second area of possible modifications is that of redundant sets. Here, however, the issue is less cut and dry. When examining the initial schema (Figure D.4) for redundant sets it becomes obvious that not only do redundant sets exist but also that some logically essential sets are missing. Specifically, it is very strange that there is no way for a customer to find out what shipments (SHIP-NO) belong to him, except via a full list of his packages (PACKAGE-NO). It is also strange that it is impossible to find out which shipments (SHIP-NO) are assigned to a given trip (TRIP-NO) except by reading all the SHIP-NOs. A SHIP-NO is an important link in the chain of relations: 1) it is via SHIP-NO that a given package (PACKAGE-NO) is assigned to a TRIP-NO; 2) it should be that a CUSTOMER works through SHIP-NO to locate his packages (PACKAGE-NO) and not the

other way around.

To incorporate these features into the system requires the addition of two sets: a 1:M from CUSTOMER to SHIP-NO, and a 1:M from TRIP-NO to SHIP-NO (Figure D.6). By so doing, the schema (the information model) becomes a more accurate picture of the inherent information structure underlying the system, and the process of searching out redundant sets can continue.[1]

There is no problem recognizing the redundant sets (listed in Figure D.7), but the problem lies in which to remove. When, for example, looking for all the products a customer has ordered should it be necessary to always trace from CUSTOMER to SHIP-NO to PACKAGE-NO to PRODUCT? Would it not be simpler to retain the redundant direct relationship between CUSTOMER and PRODUCT? Such questions can be meaningfully answered by an analyst working in a real environment. The conceptual schema is a tool, not an abstract theoretical construct, if it is more meaningful for the analyst to

---

[1]If one does not accept this analysis, or if one thinks it prejudicial to the canonical design's case, the comparison between the resultant schemata can take place either using the initial schema (Figure D.4) or using the initial schema modified to account for shared attributes and set redundancy (but without the benefit of the above added sets (Figure D.9).

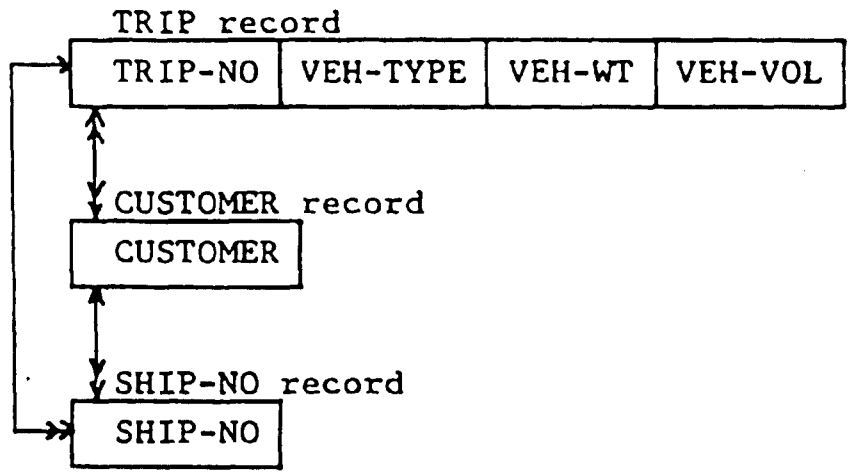


Figure D.6 Schema Modifications

Redundant Set	Non-Redundant Alternate Path
SHIP-NO → DATE	SHIP-NO → TRIP → DATE
PACKAGE → DATE	PACKAGE → SHIP-NO → TRIP → DATE
CUSTOMER → PACKAGE	CUSTOMER → SHIP-NO → PACKAGE
CUSTOMER → TRIP	CUSTOMER → SHIP-NO → TRIP
CUSTOMER → PRODUCT	CUSTOMER → SHIP-NO → PACKAGE → PRODUCT

Figure D.7 Redundant Sets

think in terms of a direct redundant link, then the redundancy can remain. Conversely, if the analyst feels more comfortable working with no redundancy, then all redundant sets should be removed. For the sake of this example, all redundancy will be removed (remembering, of course, that it can be restored as the need arises). The final schema is shown in Figure D-8.

Having derived this schema, the time has come to compare it to the product of the canonical design. Even a casual observer would note that the two schemata differ considerably; how is one to judge between them? Although it is possible to argue that one schema seems more flexible than the other, or that one seems less complex than the other, the first test should be how well each satisfies the needs it was created for. If a schema does not conveniently support the applications it was designed around, then there is no point in discussing flexibility, complexity, or any other issue.

In this comparison each user's view (or local view) will be listed and each schema examined to determine how that view can be supported. The supporting commentary for the canonical design is that of Raver and Hubbard (1977, pp. 296-297).

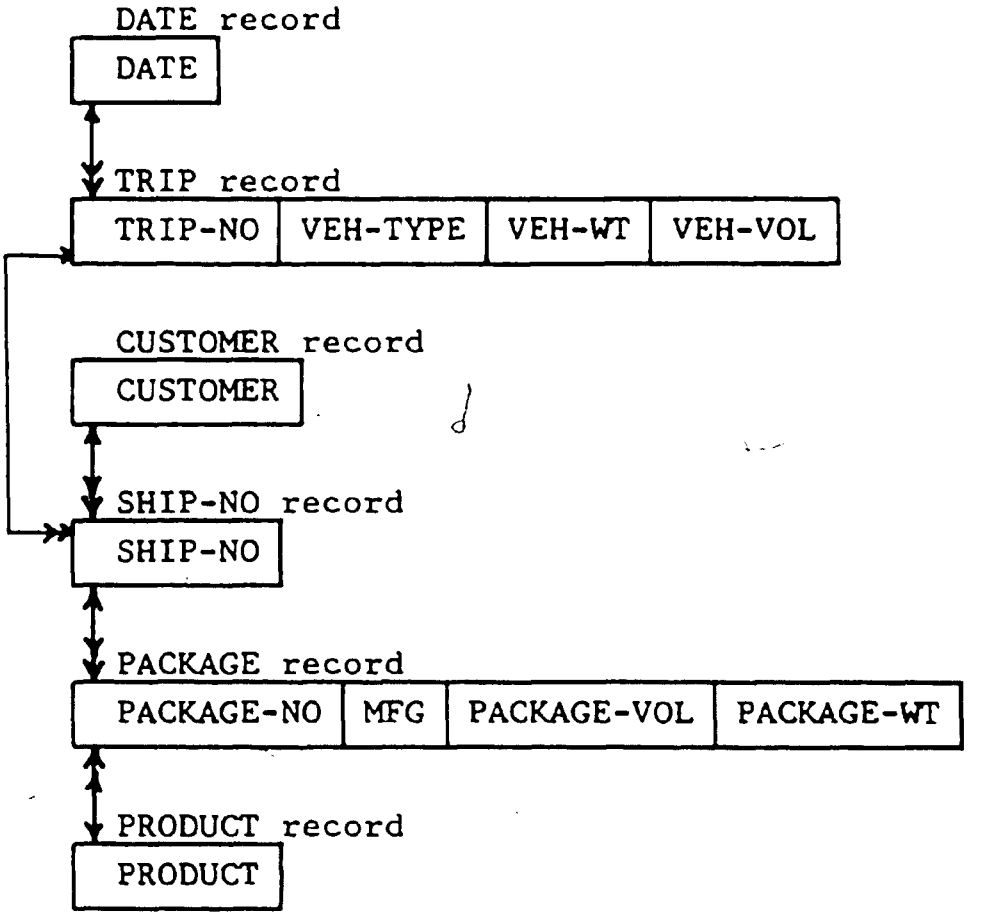


Figure D.8 Eidos based Schema

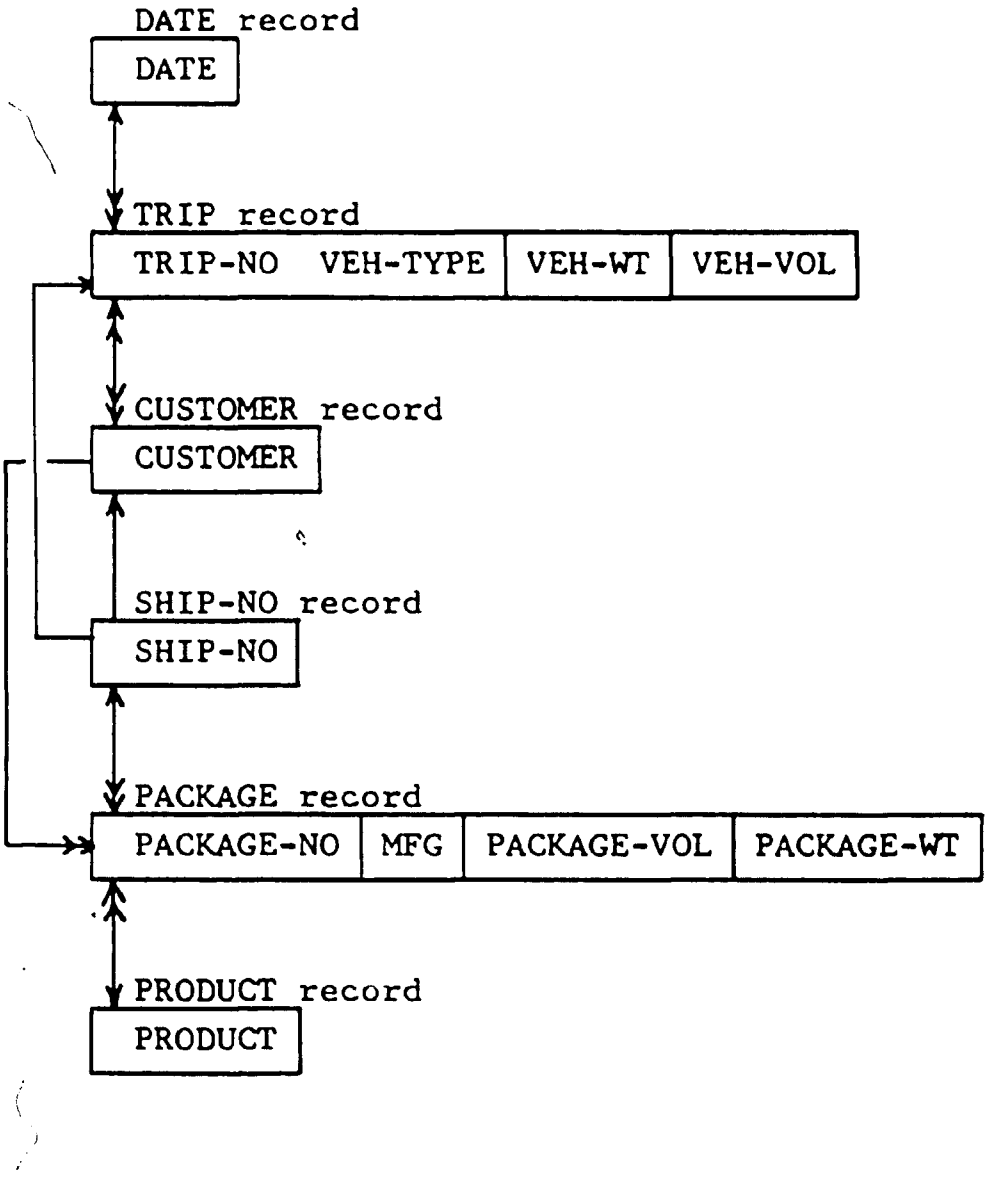


Figure D.9 Alternate Eidos based Schema

View #1

"the trip schedules . . . that list each trip by date, and for each trip, give vehicle type, weight, and volume."

- the canonical design: "View 1 can be satisfied provided a secondary index is implemented with DATE as a source and TRIP-NO as target."

- the eidos design: View 1 is directly supported by the schema, there is a direct link between DATE and the TRIP-NO record which contains all the requisite information.

View #2

"The customer shipment query . . . handles queries about the dates of scheduled trips to customers."

- the canonical design: "View 2 requires a secondary index with CUSTOMER as source and TRIP-NO as target."

- the eidos design: View 2 can be accommodated by working from a CUSTOMER through his SHIP-NOs to the appropriate TRIP-NOs (and their respective DATEs). It should be noted that if it is considered important to have a direct relationship accommodate this view, the redundant relationship between CUSTOMER and TRIP-NO need not have been eliminated.

View #3

"the customer product query . . . such as, 'When and what is the shipping information for given products?'"

- the canonical design: "View 3 is not efficiently supported by the canonical representation. One and possibly two sorts will be required to produce the report. A secondary index with CUSTOMER as source and SHIP-NO as target will avoid an additional sort. The designer may wish to reconsider and modify View 3 . . . to avoid the sorting."

- the eidos design: This view can be accommodated with the schema by connecting CUSTOMER to PRODUCT through PACKAGE-NO (which must be accessed anyway), and by connecting PACKAGE-NO to DATE through SHIP-NO (which also must be accessed anyway). Once again, if any of these connections were deemed important enough to warrant the existence of a redundant set the appropriate set(s) could have been spared in the elimination process.

View #4

"The trip contents . . . lists each trip, the customers to be served, and the packages and products to be delivered."

- the canonical design: "View 4 is directly supported by the canonical representation."

- the eidos design: View 4 is supported by connecting CUSTOMER to TRIP-NO through SHIP-NO and then direct links from SHIP-NO to PACKAGE-NO and from PACKAGE-NO to PRODUCT.

View #5

"The shipment history . . . provides a history of each shipment."

- the canonical design: "View 5 requires a secondary index on SHIP-NO and a backward pointer from the SHIP-NO segment /record/ to the TRIP-NO segment."

- the eidos design: View 5 "is directly supported by" the eidos approach.

There is little that need be added to the conclusiveness of the above comparison. Whereas the canonical design schema has problems accommodating most of the user's views (and has to resort to secondary indicies and sorts in order to do so), the eidos design directly supports them all. The only aid which the eidos design needs is the occasional use of an intermediate logical connection (which it should be pointed out, follows intuitive lines of thought - such as connecting a customer to his packages via a shipping number).

VITA

Kerry Nemovicher was born 29 September, 1946 in New York City. He graduated from Roslyn High School in June 1964 and began attending St. John's College (Annapolis, Maryland) in September of the same year. St. John's College is a classical Liberal Arts institution based upon the "Great Books of the Western World" program. Although the St. John's program is highly structured and academic, Mr. Nemovicher's extra-curricular activities were many and varied. During the academic seasons he participated in community affairs (local politics and a volunteer program at the state mental health institution) and student government. Over the summer vacations, Mr. Nemovicher built cabins for a homesteader in Alaska, re-wrote the St. John's freshmen laboratory manual and studied archeology at Oxford (England). In June of 1968, Mr. Nemovicher graduated from St. John's College (B.A. Liberal Arts). His bachelor's thesis topic was "Mitzvah: A Judaic Interpretation of the Good Deed."

In August of 1968 Mr. Nemovicher immigrated to Israel taking up residence at Kibbutz Mishmar HaEmek. He worked there (mostly agricultural work) until November 1969 when he was inducted into the Israel Defense Forces. At the end of the three year tour of duty he began a career as a computer programmer. As a key member in a "technical

control group" in the data processing center of one of Israel's largest computer installations, he gained experience in both applications and system programming, as well as organizing instructional programs. At the Israel 10th Annual Data Processing Conference (in Jerusalem) he delivered an original paper: "COMFORT - A Report Generating Language".

In the fall of 1976, Mr. Nemovicher began his studies in the Industrial Engineering department of Lehigh University.