1-1-1983

Data semantics, data modeling, and their application to the management of geopolitical statistical data.

Stuart A. Westin
University of Massachusetts Amherst

DATA SEMANTICS, DATA MODELING, AND THEIR APPLICATION TO THE MANAGEMENT OF GEOPOLITICAL STATISTICAL DATA

A Dissertation Presented

By

Stuart A. Westin

Submitted to the Graduate School of the
University of Massachusetts in partial fulfillment
of the requirements for the degree of

DOCTOR OF PHILOSOPHY

May 1983

School of Management

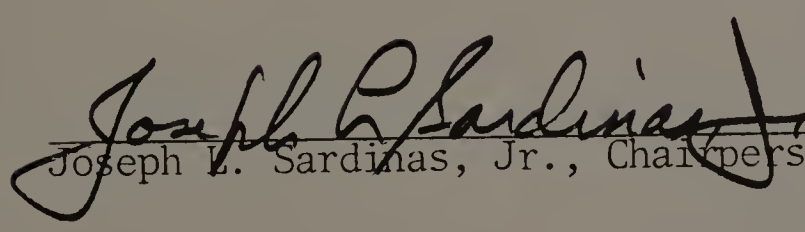
DATA SEMANTICS, DATA MODELING, AND THEIR APPLICATION TO THE MANAGEMENT OF GEOPOLITICAL STATISTICAL DATA

A Dissertation Presented

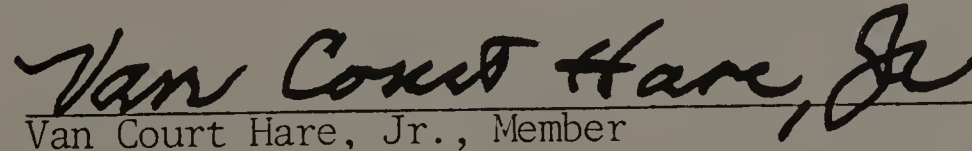
By

STUART A. WESTIN

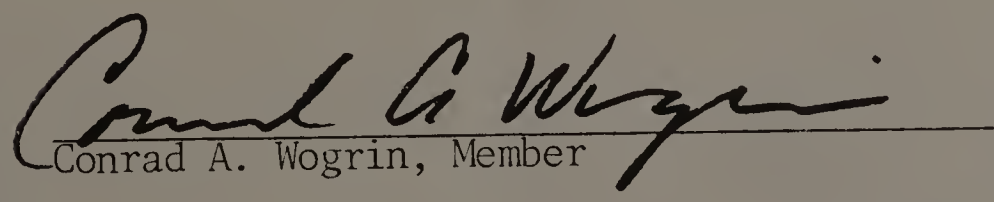
Approved as to style and content by:



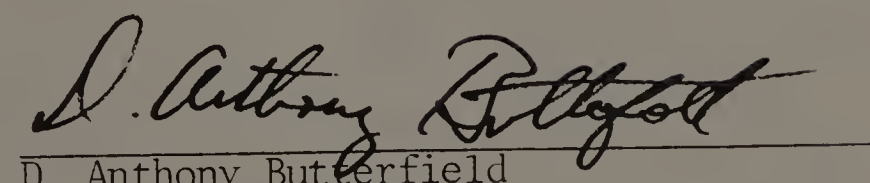
Joseph L. Sardinas, Jr., Chairperson of Committee



Van Court Hare, Jr., Member



Conrad A. Wogrin, Member



D. Anthony Butterfield
Ph.D. Program Director
School of Business Administration

DEDICATION

This dissertation is lovingly dedicated to the memory of my father, Waldemar Westin, and to the memory of my mother, Helen (Lifner) Westin. They will remain on my mind and in my heart always.

ACKNOWLEDGEMENTS

With any work of this size there are a number of people whose help and encouragement deserve acknowledgement. This dissertation is no exception. Indeed, when I consider all of my friends and colleagues who, either directly or indirectly, helped to make my doctoral experience bearable, I realize that attainment of my degree is a collective rather than solitary achievement. I include in this group not only those persons who assisted me in my dissertation work, but also those who helped me to "clear the hurdles" -- core exams, course work, comprehensives -- of my doctoral program. Without their help I might never have reached this final dissertation stage of my program (Phase III in the administrative jargon of the doctoral studies office).

First of all, I would like to thank Joe Sardinas, my dissertation committee chairman and mentor, for his valuable help with my dissertation research, with my comprehensive exams, and with my doctoral course work. I hope that in the future I will prove to be as supportive and as helpful in dealing with my students as Joe has been with me. Thanks Joe, you are a valued friend. Oh, by the way, did you hear the one about the....

I would also like to take this opportunity to thank Van Hare and Conrad Wogrin, the other members of my committee, for their help with this dissertation project. Each of them contributed a generous amount of knowledge and expertise to this work, and their inputs were invaluable in its completion.

Van Hare was especially helpful in providing greatly needed editorial suggestions. I must admit that the chapter structure of this dissertation is largely based on Van's valuable inputs.

Conrad Wogrin was always willing to take time from his busy schedule as Computing Center Director to advise me on this project.

To my close friend Mike Becker, Computing Center Information Systems Director, I express my gratitude for sharing with me his ideas on aspects of the early version of my modeling system. Undoubtedly, Mike deserves credit for defining the initial problem addressed in this dissertation and for spawning my interest in this particular data management issue. Thank you Mike.

George McDowell, director of the Center for Massachusetts Data, provided a great deal of help in defining the needs of the data users. George was also responsible for providing Center for Massachusetts Data funds to aid me in my early development work.

Professors David Stemple and Bruce Croft, both of the Computer and Information Sciences Department, instructed much of my study of database management and data modeling. Each of these fine professors, on a number of occasions, took the time to address my personal questions. For this, they have my thanks.

The members of my doctoral program cohort -- Stella Nkomo, C.N. Hetzner, Marty Moser, and especially Mike Moore -- deserve thanks for the friendship, advice, and support they provided during our years of doctoral study. As academic colleagues these people were never averse to sharing their thoughts and ideas. As dear friends they never failed to provide that undefinable yet invaluable gift that friends give.

Still other friends who were not directly involved in my studies merit thanks for making my years of doctoral work tolerable. My brother Bill was at all times willing to give advice and encouragement. Harry Tamule and Tedd Brown, who have grown to be two of my closest friends over the past few years, were unselfish in accepting my many moods during the difficult periods of my course of study.

Finally, I wish to thank Joan Ryan and Judy Sugrue for their expert typing assistance, and Despina Varelakis for her meticulous proofreading and editing of the final

draft. Joan and Judy are perhaps the only people in the world, besides myself, who have learned to interpret my near illegible handwriting. Despina is undoubtedly the only person who will ever consider in detail every single word, space, and mark of punctuation in this lengthy work.

To all of you, again, a hearty and sincere THANKS.

Postscript. After completing the dissertation "front material," of which the above acknowledgement section is a part, I proceeded with the job of assembling the final copy of the diagrams and figures. You can see that there are quite a few of these, ninety-eight to be exact, and virtually all of them required that labels be typed within little boxes, triangles, circles, or what have you. I was quick to learn that this typing chore is not as easy as it sounds. My only saving grace was the "auto-correct" feature of the IBM typewriter which I was using (thanks to Lou Wigdor). I am sure that this ingenious mechanism alone saved my sanity in completing the anguishing task. So, to the persons involved in the development of this feature -- whoever and wherever you are -- I give my lifelong gratitude.

ABSTRACT

DATA SEMANTICS, DATA MODELING, AND
THEIR APPLICATION TO THE
MANAGEMENT OF GEOPOLITICAL
STATISTICAL DATA

(May 1983)

Stuart A. Westin, B.B.A., University of Massachusetts
M.S.B.A., University of Massachusetts
Ph.D., University of Massachusetts

Directed by Dr. Joseph L. Sardinas Jr.

This work is in three major parts. In part I we provide an extensive discussion of data semantics, of conceptual data modeling, and of data modeling formalisms.

In part II we consider the nature of geopolitical statistical data, that is, statistical or summary data which is geographic based. Improvement of the application environment of data of this type is the ultimate focus of our efforts. In addressing this issue it is decided that said environment deserves and requires its own special purpose modeling formalism. To this end we develop the Geographic Entity-Relationship Modeling System (GERMS) -- a data modeling formalism through which logical models, or conceptual schemas, of geographic based statistical data sets can be created. The GERMS formalism in and of itself provides a medium through which data set logical contents can be (1) documented, and (2) discussed.

In part III we design a GERMS-based computer information system. Here, the modeling formalism is made an integrated part of a DBMS thereby giving the system the ability to "comprehend" references to a GERMS data model. These references are made in terms of our system queries, the structure of which is designed with user needs in mind. Since the system can comprehend references to the data model, it has the ability to verify the semantic (not just syntactic) correctness of the users' queries. Thus, the system can insure that queries "make sense."

A number of system enhancements are also developed. One such enhancement is a semantic integrity checking mechanism which awards the system the ability to verify data integrity according to data meaning. A second is a system merging facility which aids in the blending of two GERMS-modeled databases. Other system enhancements considered are a user view facility and an associative query handling capability.

An extensive appendix illustrates how the methodology developed is easily implemented within an existing DBMS. The SIR version 2 system is used for this purpose.

TABLE OF CONTENTS

PART II

THE NATURE OF GEOPOLITICAL STATISTICAL DATA -- STRUCTURE AND SEMANTICS

PART III

A GEOGRAPHIC INFORMATION SYSTEM

APPENDICES

LIST OF ILLUSTRATIONS

C H A P T E R I

INTRODUCTION

In recent years the terms "database management" and "database management system" (DBMS) have been used with increasing frequency in data processing books and journals. The work hereunder deals with a closely related issue -- data modeling. Specifically our interests lie in data modeling of the semantically rich, or "conceptual," variety. Our concerns are directed towards the geopolitical statistical data management application environment. Our ultimate goal lies in the improvement of the conceptual modeling of such applications.

By geopolitical statistical data we mean those data sets in which summary or statistical data is geographic based. An example of such a situation is found within the contents of U.S. Bureau of the Census supplied "summary tape files" (STFs). Here, data observations are aggregated and provided according to a number of well defined partitionings of the geographic territory from which they are collected (see [KAPL80]). There is no means by which the data can be disaggregated, thus each datum within the data set is a statistical object which is "pinned" to some geographic area (vis. geographic point).

The applications of data of this type are most often of a geopolitical nature. The data are used as the foundation of geographic or geopolitical database systems. The decision makers that are served by such systems may be business executives, government officials, socio-demographic researchers, and the like.

Since the majority of such database systems are founded partially or totally on Bureau of the Census supplied summary tapes, the geographic issues and examples discussed in the work will relate closely to census data sets. The concepts are generalizable to non-census geographic database systems, however.

We will begin the discussion with an investigation as to the present state of said environment: "It is usually the case that business executives, government officials and other decision makers have a good idea of the kind of information residing in their databases. Yet to obtain the answer to a particular question, they generally need to employ the services of a technician who works with the database on a regular basis and who is thoroughly familiar with its file structure, the DBMS on which it resides, how it is distributed among various computer systems, the coded field names for the data items, the kinds of fields that the different files are expected to contain, and other idiosyncrasies.

"The technician must understand the decision maker's question, reformulate it in terms of the data that is actually stored, plan a sequence of requests for particular items from particular files on particular computers, open connections with remote sites, build programs to query the remote systems using the primitives of the DBMSs of the remote systems, monitor the execution of those programs, recover from errors, and correlate the results. This is a demanding, time-consuming and exacting task requiring much attention to detail. Escalated levels of sophistication are needed as the VLDB [very large database] increases in size and complexity and as it is distributed over a wider range of host computers" [HEND78, pp. 105,106].

When a data model is implemented it frees the user from the need to deal with many of the complexities of these technical issues; it allows the user to interact with the database "in terms of" the logical contents of the data set. As stated above, our work here deals with the creation of data models which are dedicated specifically to the needs of the users of geopolitical statistical data.

The logical contents of a data set can be defined in terms of four basic objects: entities, associations, properties, and values. The latter three are important only because they are used to describe the entities. We can describe an entity by asserting that it holds some value relating to a property. Alternatively, we can describe an entity by asserting that it is associated in some way with another entity.

A geographic based data set is no different. It can be portrayed in terms of entities and descriptive assertions about these entities. What is unique about this environment is the nature of these objects. The entities of a geographic data model -- those things about which assertions are made -- are geographic areas. Thus states and counties are example entity types.

As is the usual case, assertions are of two general categories. One type of assertion which can be made about an entity has to do with properties of the entity. The properties hold values. In the geographic oriented data modeling situation these values exist as statistical summary objects. For example, by recording a population datum within a township data record we are asserting that an entity (township) holds a certain value on some property (population).

It must be noted that, in this application environment, entities are partitionings of other entities. Properties, then, represent statistical measurements based on different partitionings of entities. Such a situation is rarely, if ever, found in other data management environments.

The second category of assertion, assertions about associations among entities, deals with the spacial overlap of the objects. Geographic entities are related when they share a common area. For instance, a given township is related to a given county if the township lies within the county. Otherwise, the objects are unrelated. Although the semantic importance of such relationships among geographic entities cannot be denied, we see them as being "passive" relationships. This passivity is cited in opposition to the more "active" relationships found in traditional (e.g. business) database applications such as "works for," and "manages."

Consideration of the relationship of an individual pair of geographic entity types, as in the example above, is trivial. If all geographic data sets were based on county and township partitionings only, the management and access of such data would be simplistic. In reality, however, these tasks are complex.

There are three main reasons for this complexity. First, a geographic data set is often based on many different geographic partitionings. As a result, we must simultaneously consider many geographic relationships.

Second, the set of geographic types upon which the data are provided vary from application to application. Consequently, relationships which exist within one data set are often quite different from those that exist within another geographic data set.

Third, and perhaps most important, the geographic entity types upon which a data set is specified are often defined according to different criteria. The result of this fact is that the geographic types are not compatible in terms of their boundary definitions.

In census data sets geographic partitionings are of three general categories. Geographic entity types may be statistical, political, or enumeration areas (the latter are sometimes considered to be statistical areas). Figure 1.1 lists some of these geographic area types. A more comprehensive list is found in appendix B.

All data sets, including geographic based data sets, are bound to some physical structure. In massive applications, such as those discussed here, physical data structure is designed to meet the requirements of

Geographic Areas

Census data are presented for various political and statistical areas.

Political areas include:

United States

States (and outlying areas)

Congressional districts

Counties

Minor civil divisions: Legal subdivisions of counties, called townships in most States

Incorporated places: Cities, villages, etc.

Statistical areas include:

Census regions and divisions: The 50 States have been divided into four regions, each containing two or three divisions.

Standard metropolitan statistical areas (SMSA's): Usually consist of a central city with a population exceeding 50,000, the county(ies) in which it is located and other contiguous counties that are metropolitan in character and are socially and economically integrated with the central city.

Urbanized areas: As defined by population density, each includes a central city and the surrounding closely settled urban fringe (suburbs) which together have a population of 50,000 or more.

Urban/rural: All persons living in urbanized areas and in places of 2,500 or more constitute the "urban" population, all others constitute the "rural" population.

Census county divisions: Statistical subdivisions of a county defined for those States where minor civil divisions are not appropriate for the publication of statistics.

Census designated places: Residential concentrations related to a geographically defined "place," although the "place" is not legally incorporated.

Census tracts: Statistical subdivisions of an SMSA with an average population of 4,000.

Enumeration districts: Census collection areas used as tabulation areas where block statistics are not collected.

Block groups: Census tabulation areas intermediate between census tracts and blocks.

Blocks: The smallest census geographic areas, used as basic tabulation areas in urbanized areas and incorporated places with a population of 10,000 or more.

ZIP code areas.

SOURCE: [KAPL80, p. 104]

FIGURE 1.1

efficient data processing and storage. This physical structure seldom reflects the logical structure or conceptual meaning of the underlying data.

The response of the data management community to this issue is to provide, for each application, a logical model (conceptual schema) which is independent of any physical structural considerations. In the jargon of database management there exists "physical independence" of the logical and physical structures. The advantages of providing a conceptual model of an application are realized in both the utilization and the administration of the database.

At present there does not exist a formalism which supports the creation of adequate conceptual models of geographic databases. This deficiency is due to the unusualness of the involved database objects. Existing modeling techniques are not well suited to model the semantics of spacial overlap or statistical properties.

A result of this indigence is that geopolitical database usage and administration revolves around the physical data structure only. Furthermore, all documentation is tied to physical structural issues with little regard for the conceptual structure of the database.

We do not mean to imply that the contents of the geographic database are misunderstood by all who use them. Obviously, the database administrator(s) and technicians, through experience and the passage of time, gain a comprehension of the semantics of the database contents. This "meaning" is embedded in the knowledge of the technicians and in the database application programs, however. Also, it is difficult to verify that the understanding maintained by a database technician is consistent with reality or with that of other technicians and administrators.

The present situation gives rise to a number of problems in terms of data access and usage by the naive or occasional database user (called "casual user" in database literature). First, since the data are provided according to a number of non-compatible partitionings of geography, the objects of the physical data structure need not be logically meaningful. All database documentation and description is based on physical structure so, consequently, the user is forced to contend with confusing and unnatural data objects in conceptualizing the database contents. There exists no concise list of the logical contents of the system.

Second, once the individual geographic entity types are known, the meaning and interrelationships of the geographic types must be assimilated. This intricate task is presently accomplished by studying glossary type definitions of the individual objects and by deriving pair-wise associations from this information.

A related issue is the fact that, once the entire set of geographic associations is comprehended, some subset of geographic object types upon which data access will be based must be selected and respective interrelationships understood. Alternatively, the user is forced to rely on the aid of a (hopefully) knowledgeable technician.

The database technician who tutors the potential user about the logical structure of the geographic information system is faced with the problem that natural language is the only existing communication medium for such a description. This medium of discourse is hardly well suited to address the complexities of geographic semantics. Still, an alternative modeling technique does not exist.

It is these problems (and some surrounding issues) which are addressed in this work. The specific topics covered are detailed below.

What Lies Ahead

Beyond this introductory chapter our work is composed of three major parts. Part I, which comprises five chapters, is aimed at providing for the reader a technical foundation in the area of conceptual data modeling and the issues related to such. Much of this foundation is based on concepts found in relevant literature in the areas of database management, data modeling, artificial intelligence, and of course the comprehensive area of computer science.

In chapter 2 we take a "generic look" at data modeling. First, we develop a notational form which can be applied in the description of any data model. Next, we consider the three "dimensions" of a data model. Specifically, these are constraints, structures, and operations.

Following this, we discuss the ways in which data models and data modeling formalisms (DMFs) can be evaluated and compared. Not only do we consider how instances of these can be compared to one another, but also how they compare, in general, to topics with which we are much more familiar -- programming and programming languages.

Finally in this chapter, we consider the concept of the generations of modeling formalisms. We make the distinction between the "traditional" first generation formalisms and the second generation "semantic DMFs;" the latter being defined as the primary focus of the subsequent chapters.

Data semantics, the concept which underlies second generation data modeling, is the topic of chapter 3. This chapter has three major sections. First, we attempt to gain an understanding of semantics by contrasting it to syntax, its less elusive complementary counterpart. Second, we describe a theory of data semantics and data modeling. Third, we investigate the treatment of the topic of data semantics within the literature of database studies.

In chapter 4 we discuss the fundamental objects with which a data model deals. We will learn that the fundamental "primative" objects of a model are combined to form groups, called types, which themselves are fundamental objects of the data model. A number of different approaches to the typing process are considered.

Chapter 5 takes this notion of object groups one step further by discussing data abstraction. This concept, which is basal to the study of conceptual data

modeling, considers the ways in which the above noted types (groups of primitive objects) can be further grouped in the formation of even more data model objects. We will see that data abstraction is of two distinct kinds: aggregation abstraction and generalization abstraction. Each of these kinds is considered in detail.

The final chapter of part I, chapter 6, addresses a number of issues which relate to the implementation of a data model. Here, we relate the structures of data models to the actual data sets whose contents they depict.

The text of appendix A complements the topics of part I in that it describes a number of basic schools of and approaches to conceptual data modeling. In addition to detailing several modeling formalisms which have developed from database management research, this appendix also describes the two very different approaches which are subscribed to by the artificial intelligence community.

The treatment of topics in part I is extensive since we feel that meaningful research ideas deserve and require a solid base on which they may be built. The practitioner who is only marginally interested in these issues is urged, at least in the initial reading, to omit part I and begin directly with part II.

In this second part, which comprises three chapters, we investigate the nature of geopolitical statistical data sets in terms of their structure and their semantics. In chapter 7 we focus on the physical structure of such data sets. This begins with a brief review of tree structures and hierarchical files since the data sets with which we are concerned are physically configured according to these structures.

Next, we discuss the conceptual anomalies which accompany the use of a hierarchical physical structure as a model of the logical data contents. We will learn that hierarchies, while possibly serving adequately as a physical data framework, are hardly well suited for the task of conceptual representation.

Having realized the shortcomings of the hierarchy as a conceptual modeling tool, we focus in chapter 8 upon the development of a data modeling formalism which is better tailored for the situation at hand. This formalism, which we call the GERMS (Geographic Entity-Relationship Modeling System), allows for the creation of a "picture" of the logical contents of a geopolitical statistical data set. There is provided for the GERMS a graphical as well as a lingual form. Both forms produce equivalent data models.

Appendix B, which provides a detailed glossary of census geographic terms, should prove helpful in reading this chapter.

The final chapter of part II, chapter 9, reconnoiters and evaluates the GERMS formalism in terms of a number of concepts described in part I. As the focus of this discussion is academic as opposed to pragmatic, the practitioner is advised to omit the chapter in the initial reading.

In part III we consider the "machine implementation" of GERMS data models. The system which is designed here has the ability to respond to unplanned, or ad hoc, information requests. Furthermore, these requests, called queries, are posed to the system "in terms of" the GERMS data model. Because of these features we call the system a geographic information system.

Part III comprises the final five chapters of the work. In chapter 10 we consider the form that the above noted queries should take on. Naturally, this form is considerate of the needs of the system user.

Chapter 11 focuses on the design of the physical database of our system. The first concern addressed here is that of relating the existing physical data structure (described in chapter 7) to the objects of our logical

data model. The treatment of this topic within the physical design chapter may seem out of place, but we will learn that the ultimate result of this disposition is a better understanding of how the physical data should be structured.

In the remainder of the chapter we describe a revised physical data structure which is better suited to handle the needs of processing ad hoc GERMS-based information requests.

In chapter 12 we deal with the system logical database. The logical database holds an internal (to the system) representation of the GERMS data model. Also, it retains the information that is required to "link" this data model representation to its physical data structural counterpart. Both of these aspects of the logical database are detailed. In addition, we discuss how the logical database contents are "loaded" into the system. This involves the use of a data definition language which is based on the aforementioned GERMS lingual form.

In chapter 13 we consider a number of "amenities" which can be used to refine the GERMS-based system. The first such enhancement has to do with the ability to handle associative queries. We use this term to refer to information requests which are based on (non-key) attribute values.

The second amenity deals with the maintenance of semantic integrity within our database. We develop a semantic integrity checker which verifies, automatically, the integrity of data value according to the "meaning" of the data. Obviously, this data meaning is extracted from the data model which is stored within the logical database.

The third system amenity is a facility which aids in the merging of two conceptually modeled databases. With this software based merging facility the structures of the GERMS data models are used to direct the combining of two data sets which represent different properties of geographic areas. Following this merging, software is used to "generate" geographical statistical data which previously (i.e. before the merging) did not exist in either of the original databases.

The fourth and final amenity discussed here has to do with the use of user views of the (global) GERMS data model. Through implementation of this user view concept each application which the information system serves can utilize its own dedicated version (view) of the data model. Such a user view is ignorant of all logical objects with which the application is not concerned.

Appendix C illustrates how the methodology described in part II and part III is implemented using an actual database management system. Specifically, the SIR (Scientific Information Retrieval) version 2 system is used. As the methodology is software intensive, a significant portion of this appendix presents and describes programs which are written in the SIR language.

Chapter 14, which is the closing chapter, holds a brief review and the concluding remarks of this work.

PART I

AN OVERVIEW OF CONCEPTUAL DATA MODELING

". . . I have only made here a collection of other people's flowers, having provided nothing of my own but the cord to bind them together."

Montaigne
"Essays"

C H A P T E R I I

A GENERIC LOOK AT DATA MODELING

It would be difficult to argue against the need for database technology in today's society: "Social changes have created large and complex political and economic structures (e.g. large government agencies, multinational corporations, and chains of retail stores). These enterprises need to integrate the use of their data for planning and control. Database management systems are an appropriate tool for this integration. The main issue is effective use of data rather than the particular system that is used for their storage" [TSIC82, p. x].

Of paramount importance in designing a database is selection of some set of constructs through which the contents of the database will be expressed. This has been cited as being the most important problem related to the database design process [WONG77]. Such sets of constructs are often called "representations" in artificial intelligence (AI), and "data models" in database management (DM) literature.

It is our belief that use of the term "data model" is incorrect in this context. A model is the result of application of (and adherence to) the set of constructs

rather than the set of constructs itself [1]. Consequently, we will refer to such sets of constructs as data modeling formalisms (DMFs). The process of applying the constructs will be called data modeling. A paradigm that is created as the result of said process will go by the title data model.

It is this realm of database technology, the study of data modeling and of modeling formalisms with which we are concerned. The obvious way to begin our discussion is by presenting a formal notation and set of definitions so that a generic undersanding of data modeling can be achieved. The concepts presented below show close adherence to those presented in [TSIC82, sec. 1.3].

A Notation for Data Models

A given DMF has two major components: (1) a set of generating rules (G), and (2) a set of operations (O). The realization of G shows itself in terms of a formal data definition language (DDL). The purpose of G is to define the structure of objects as they appear in the database.

The role of these structures is twofold. First, a structure is imposed upon objects through specification of their componential makeup and categorization. Second, structure is imposed upon sets of objects through specification of the ways in which they may be associated. The set of operations relates to the data manipulation language (DML).

In terms of representing the real world, G captures the static properties; those general characteristics of the world that always hold. The operations allow for the representation of real world dynamics.

The generating rules are used to define a schema s:

$$G \rightarrow s$$

It is often useful to think of G as being composed of two parts: (1) a structure part G(s), and (2) a constraint part G(c). Our schema is partitioned accordingly:

$$G(s) \rightarrow s(s)$$

$$G(c) \rightarrow s(c)$$

The constraint schema is used to specify the explicit constraints that are to be imposed upon the database by clearly defining which database conditions are allowed. Note that s(s) often includes some implicit constraints that are defined by the nature of its structure. If the structural specification of s(s) is to be adhered to, certain states are eliminated from the realm of possibility.

A given G defines the range of S: the set of all possible schemas that can be derived from G. One instance s of S allows for the realization of a set of permitted database occurrences (D) all of which obey the generic rules G of DMF.

Each primitive operation o of the set O provides a transformation from one database state (dbs) to another database state such that dbs(i) may violate s(c).

$$o: dbs(1) \rightarrow dbs(2)$$

The elements of O are combined to define transactions.

Performance of a transaction (t) yields a transformation from one database occurrence (d) of the set D to another database occurrence of D.

$$t: d(1) \rightarrow d(2)$$

In this way the dynamic properties of the real world are emulated.

Since the pre and post transaction conditions are elements of the set D, the total transaction t is integrity preserving; it maintains the rules of G as specified by s(s) and s(c). During the performance of a transaction the database is allowed to temporarily violate s(c) as it passes through its various database states. The inherent constraints of s(s) are never violated, however.

Constraints, Structures, and Operations

The above discussion of DMFs and their utilization was tripartite. Formalisms were described in terms of their structures, their constraints, and their operations (processes). Such a partitioning of constructs is common within the DBMS literature (see [LUM78], [SU79], [TSIC82]). This orthogonal treatment should not be taken to imply that the constructs are disjoint, however. Rather, a DMF results from their coalescence.

As an analagous example consider the stack as an information structural representation. When we investigate the nature of its "stackness" we find that structures, constraints, and operations, taken separately, do not describe a stack. Instead, it is defined by the harmonic interaction of these.

In terms of structures, the stack is a simple linear list. This exact structural form is also taken on by queues, dequeues, and arrays. Therefore, it is not the structure of the stack from which its total meaning is derived. Note that stack structure corresponds to s(s), the structural schema, of our DMF specification.

As for stack operations, the "reasonableness" of performing POP and PUSH on a stack is the result of stack structure: a linear list. Tree based operations such as "preorder traversal" and "suffix walk" (see [HOR076], [PAGE78]) make no sense on such a structure; a linear list imposes a set of implicit constraints. Correspondingly, the operations POP and PUSH are nonsensical when applied to a tree.

In a similar way, the form that is taken on by the elements of O is designed for use in conjunction with the structure of s(s) as generated from G(s). The structures and operations of the DMF, like those of a stack, are interdependent.

Stack constraints go beyond the mere "reasonableness" of operations as determined by the stack structure. The constraints define the range of acceptable stack operations by placing restrictions on the domain of reasonable operations. For example, given the linear list structure of a stack, there are a number of operations whose performance makes sense. These operations have to do with how additions to, deletions from and access to the list are performed. By explicitly constraining the set of reasonable operations to be of the LIFO variety, a <u>stack</u> is defined as opposed to a queue, a deque, or an array.

Similarly, the explicit constraints of s(c) add to the implicit constraints of s(s) in determining the acceptability of database transactions. The three components, taken in isolation, do not define an appropriate paradigm of the real world. They must be applied together so that their synergism can be realized.

Comparing DMFs

Because structures, constraints, and operations vary among the different DMFs, there is a need for some means through which a comparative evaluation can be performed. One approach to this task, as evidenced by the relevant literature, is directed towards the formation of a "meta-model" which can be used as a basis of comparison.

Much of the work in this area involves the use of a binary data model [2]. This is due to the extreme power of such models [TSIC82]. For example, the potency of the Semantic Binary Data Model has been demonstrated in that the model was used successfully to describe itself [ABRI74]. Also, the modeling power of the binary DIAM modeling formalism has resulted in its recognition as a "unifying theory of database implementaion" [SCHN76].

A second approach to DMF comparison involves the specification of some universal set of criteria through which all DMFs can be compared and evaluated. Some preliminary criteria, as detailed in [LUM78], are as follows (be aware that the term data model denotes our concept of DMF):

1. Expressiveness (explicitness). Can all objects, constraints and processes be described in the data model?

2. Naturalness. Can the structure, constraints and processes of the enterprise being modeled be described in terms familiar to the end user of the system?

3. Implementability. Can the primitives of the data model be implemented in "reasonable" ways on existing DBMSs?

4. Simplicity. Are there too many constructs and too many different ways to specify the same thing in the data model?

5. Overloading. Are some constructs in the data model used to specify too many different things?

Upon inspection of these criteria we find that many of the concerns which arise in the design of a DMF closely parallel those of programming language design (see [DIJK72], [DIJK76], [ELSO73]). Indeed, the realms of data modeling and programming share a common conceptual framework. Since many of us are at ease when discussing the issues of programming, we will attempt to describe data modeling in terms of programming concepts. Hopefully, this will result in a better understanding of the modeling realm.

Programming and Data Modeling

In this comparison we see a programming language (PL) as roughly corresponding to the concept of a DMF (data modeling formalism). Just as there is a great variety of different programming languages, so is there a great variety of DMFs (see [KERS76]). Neither a single PL nor a single DMF can be identified as being the undisputed best in its respective domain.

Certain PLs are better suited to meet the needs of certain programming environments than are others. Each language has its strengths and its weaknesses and there are definite trade-offs involved in the design of a PL. FORTRAN, for example, holds great computational power but its file handling capabilities are clumsy. COBOL retains the opposite position. Languages such as PL/1 and ADA which attempt to satisfy the needs of all programming environments suffer from extreme complexity (see[HOAR81]).

Similarly, there are trade-offs involved in the design of a DMF. Features which are well matched to the demands of one modeling situation may result in clumsiness in another environment. Also, the inclusion of too many features in a DMF will make the modeling process overly complex [HAMM81; SU79; TSIC82]. There must therefore be found some optimum mix of constructs (note criteria 4 and 5 above).

A programming language is used to implement algorithms. The algorithm exists on its own and is unrelated to any one PL. In correspondence, a DMF is used to simulate real world phenomena which hold independent (of the DMF) stature. Consequently, algorithms as viewed in the programming realm relate to phenomena in the data modeling realm.

When an algorithm is implemented by adhering to the syntactic and semantic constructs of a PL, the process is called programming. The result of the process is the creation of a program.

Any algorithm can be programmed in a number of different ways within the constructs of a single PL. Furthermore, the implementation of an algorithm in two different PLs will result in the creation of two programs which differ in both syntax and semantics.

Analogously, when a real world phenomenon is represented by adhering to the syntactic and semantic constructs of a DMF, the process is called data modeling. The result of the process is the creation of a schema.

A phenomenon can be represented in a number of different ways within the constructs of a single DMF (recall the distinction between the set of schemas "S" and a schema instance "s"). Also, the representation of a single real world phenomenon in two different DMFs will result in the creation of two syntactically and semantically different schemas.

It is obvious from the above discussion that there can be drawn a number of conceptual parallels between the realms of programming and data modeling. Because most of us maintain our conceptualization of programming concepts

within a well formed cognitive framework, there are many advantages to be gained by describing data modeling concepts in terms of analogous programming concepts. We will therefore make use of such parallelism later in the discussion.

Despite the presence of certain similarities between the spheres of interest, a number of major differences do exist. These fundamental distinctions should be understood before further resemblance is noted. Some of these conceptual discrepancies, as noted in [TSIC82], are described below.

The first distinction has to do with the stage of development in which the logical structure of data is considered. In terms of programming, if we assume that the recommended (stepwise refinement) programming procedure is followed, logical structure of data is the final concern of the stepwise process. Algorithmic structure, on the other hand, is identified as being the primary consideration.

In terms of data modeling, the opposite situation holds. Here, the entire modeling process revolves around the logical structure of data, so such structure is of primary concern to the modeling process.

Algorithms can be viewed in two different lights within the realm of DMFs. If algorithms are seen as corresponding to database procedures, then their specification is a final stage of the overall modeling process. This priority is therefore exactly opposite that of algorithms within the programming realm.

Alternatively, algorithms can be seen as corresponding to database transactions. In this light, they are external to the entire modeling process; they fall within the sphere of model usage. This is the concept of logical independence -- the separation of schema from database interaction.

The second fundamental difference between the modeling realm and the programming realm has to do with the emphasis that is placed on data types. While both PLs and DMFs utilize a similar notion of data type, PLs are more concerned with processes than with data or data types. Data type constraints are enforced, but enforcement is not excessively stringent.

On the other hand, data typing is of the utmost importance in data modeling. In fact, a schema can be viewed as a collection of rigorously specified data types. The generic rules (G) of a DMF can be considered as an extensive type specification facility.

Finally, programming and data modeling differ in the emphasis that is placed on the sharing of objects. Programs seldom share (physical) data objects or variables and the specification of this within a PL is often rigid (e.g. FORTRAN "COMMON" and "EQUIVALENCE", and PL/1 "DECLARE").

In contrast, the sharing and integration of objects is a fundamental precept of data modeling. A data model shares tokens among different types and provides a number of different views through which a common set of objects can be accessed. These concepts will be expanded at a later point in the discussion.

The Generations of DMFs

Traditional DMFs can be categorized into three basic types: hierarchical, network, and relational. These have been called the "three great data models" [ULLM80]. A common thread which exists through these DMF types is that the constructs involved are mainly syntactic [SU79]. In recent years, research efforts have concentrated on the creation of a number of "conceptual" or "semantic" DMFs. These have been called "second generation" DMFs vis-a-vis the earlier "first generation" types [LUM78].

Second generation data models are not in all cases void of the first generation concepts. Rather, they are often extensions of first generation formalisms. Second generation DMFs are concerned with the meaning of data as opposed to the preoccupation with data access, storage, and manipulation that is exemplified by traditional DMFs.

The meaning of data with which these new DMFs are concerned has been cited as being of the utmost importance: "The fundamental property of a database is that it has an intrinsic meaning which is invariant of its interaction with users" ([MINS74] from [HAMM75, p. 27]).

In a more recent work we find: "They [first generation formalisms] have pointed the way, however, to all the advantages of data integration. This integration provides a framework for the operation of an organization. It needs as a prerequisite, however, a good understanding of the data. Data models provide a tool for providing such understanding. As such, they may be more important than a DBMS [database management system] per se. DBMSs may be very general and border on being a file system" [TSIC82, p. 14].

The labelling of second generation DMFs as conceptual or semantic formalisms should not be taken to imply that early database systems were vacuous of any

semantic component. The assignment of a meaningful name to a data object captures some minor data semantics. Furthermore, carefully designed tree and network structures, used in conjunction with proper record or segment specification, can result in the creation of a somewhat meaningful logical data model. The semantic richness of such constructs is far from adequate for use in representing complex phenomena, however [HAMM81; KENT79; ROUS75; WONG77].

In terms of hierarchical and network DMFs, much of the conceptual inadequacy stems from the fact that they are record based structures. As such, the models lack the flexibility that is required to represent some basic real world concepts.

As an example, consider the collection of similar objects into a generic category or type. The representation of this phenomenon is a fundamental requirement of a semantically precise model (this will be discussed in detail in a later section). The record based nature of these first generation models results in two anomalies in representing this basic construct: horizontal inhomogeneity and vertical inhomogeneity [KENT79]. The former relates to the situation where the tokens (elements) of a type do not share a common set of relevant

attributes. The latter situation arises when an attribute of a type does not share the same value format (domain) among all occurrences of the type.

Despite its mathematical foundation, the process of schema normalization, which is performed on first generation data models, is aimed at maintaining semantic correctness [ULLM80]. Nevertheless, the inadequacy of this process is often noted. Functional dependence, the notion upon which the entire normalization process is based, captures only a small portion of the total semantics of a database [CHEN76; HAMM75; ROUS75; SCHM75].

In the remainder of the discussion we will concern ourselves with conceptual data modeling. A modeling formalism which provides the constructs allowing for such modeling will hereunder be called a "semantic DMF."

Why Study Data Semantics?

If we are to concern ourselves with the study of semantic DMFs we must first be convinced that there is a need for such modeling. As the ultimate purpose of a data modeling formalism (DMF) is to aid in the management of information, our testimony begins with a statement as to

the value of information as a resource: ". . . the harnessing of information will be one of the basic tenets of corporate management in the 1980s. It is imperative that all of us concentrate on organizing and utilizing information more effectively and efficiently. Indeed, some business leaders and management consultants contend that information will be recognized as a resource comparable to capital and labor. I see information as management's most important resource for productivity" [CREN80, p. 15].

We see, then, that under or improper utilization of information can have detrimental effects upon the productivity of the enterprise. We should therefore strive to make access to meaningful enterprise information as uncomplicated and as direct as is possible. Such access requires that the information be in a usable form: "That's the key. Information. Gathered laboriously, harnessed and made viable and understandable for the decision-makers" [CREN80, p. 16].

Put another way, the database system from which information about the dynamic real world system is derived should provide for: (1) a concise, understandable representation of the state of the system at any time, (2) efficient and high level access to information, and

(3) simple and consistent update as the system changes state (from [SMIT77a, p. 405]).

Under traditional implementations, access to information is not direct and simplistic. Rather, it is periphrastic and tortuous: "The usefulness of DBMSs is severely restricted by their failure to take into account the semantics of databases. Although all three models [hierarchical, network, and relational] provide a logical view of the database in terms of data structures and a set of operations on them, they fail to incorporate the semantics of the database into these data structures and operators" [ROUS75, p. 145].

The indirect nature of information access from contemporary systems is due to the fact that semantics must be consciously applied by the designer and user of the system; they are not readily apparent from the schema [HAMM81]. The way people are forced to view information by a data model may shape their perceptions of the information. Consequently, data representation should match the way we think [TSIC82].

In summary, we see that the productivity of the enterprise hinges on the effective and efficient use of information. If the utilization of enterprise information is to meet these requirements it must be available in a

usable form. This demands that the underlying data from which the information is derived is thoroughly understood by the user.

Because traditional systems do not provide such an understanding of data they do not support the needs (design, evolution, use) of complex database applications [HAMM81]. The systems therefore fail to meet adequately the ultimate needs of the enterprise. Consequently, we feel that the study of semantic DMFs is a valuable pursuit.

This view is apparently shared by E. F. Codd, the "father" of the relational data model: "The goal [semantic data modeling] is nevertheless an extremely important one because even a small success can bring understanding and order to the field of database design. In addition, a meaning-oriented data model stored in a computer should enable it to respond to queries and other transactions in a more intelligent manner. Such a model could also be a more effective mediator between the multiple external views employed by application programs and end users on the one hand and the multiple internally stored representations on the other" [CODD79, p. 398].

Having completed our (hopefully successful) argument supporting the validity of our study of semantic DMFs, we focus on the question of why traditional data modeling techniques fail to provide the required semantic richness. The reason stems from the fact that such techniques separate data from its interpretation. This is not the case with other forms of communication with which we are familiar.

In natural language discourse, for example, both data and interpretation appear together [TSIC82] [3]. Thus, in the natural language phrase "His salary is twenty-two thousand dollars," we capture both data value (22,000) and data interpretation (dollar value of his salary). In regards to the meaning of the terms "his" and "salary," we may draw on the context in which the phrase is used.

Here, context or setting provides semantics in a number of ways. First, we can use the meaning that has been derived from the phrases of the discourse that were used prior to the one in question. Second, we receive audio and visual cues from the sender such as tone and inflection of voice, and facial expressions.

If this phrase were to be represented within a computer, the interpretation would be stripped from the data; only the value 20,000 would be stored. As a result, the intelligence of the software which accesses the data, or the intelligence of the user of such software is relied upon to provide an interpretation of the data. Since first generation DMFs are concerned with data syntax as opposed to data semantics, the user is awarded a large portion of said responsibility in these system implementations [HAMM81].

Some of the problems that arise from the lack of first generation DMFs to incorporate semantics into their structures and operators have been outlined as follows [ROUS75]:

1. Each user is required to know what the attributes and relations (records) mean; otherwise they are unusable

2. The concepts do not provide an adequate means for expressing the semantic relationships that may exist between items constituting the database. Also, they do not indicate how to choose a schema for a particular database.

3. Constraints on the execution of a particular database operation are related only to cost and security. They do not constrain database operations to make sense (other than syntactically).

4. Database consistency is a subjective notion. The effect of user modifications (insertions, deletions, updates) upon the database is understood only by the user in terms of his/her subjective view.

Later in the discussion we will see how second generation DMFs and related research efforts are dealing with these issues. First, however, we must strengthen our understanding of the construct which underlies these issues -- data semantics. The purpose of the following chapter is to provide this understanding.

FOOTNOTES

[1] Consider that a regression model is the product of proper application of linear regression techniques, or constructs, to a set of observations.

[2] A binary data model is one in which simple object nodes are connected by pairs of directed arcs. Each arc within a pair is the inverse of the other arc. Such a modeling formalism provides specificity at the expense of complexity (see appendix A for details).

[3] We will ignore, for the moment at least, the fact that natural language has an inherent ambiguity.

C H A P T E R I I I

DATA SEMANTICS

In this chapter we attempt to define the "meaning" of data semantics. The notion of data semantics, as we will soon learn, is an elusive one; the task at hand will not be easy. Still, an understanding of semantic DMFs (data modeling formalisms) demands a prior understanding of the construct from which they derive their name. We will therefore focus our efforts in this direction.

The Syntax and Semantics of Data Models

Semantics is a construct which is well entrenched in the realm of programming languages. We believe, as noted above, that there is a value to be gained by studying unfamiliar data modeling concepts in terms of familiar programming notions. Programming concepts is therefore where we will begin.

Data model syntax. The characterization of a program is twofold. It is defined by: (1) the syntax of the program,

and (2) the semantics of the program [LEE76]. The former relates to the grammatical correctness of the program as specified by the constructs of the language in which the program is implemented. The latter, semantics, has to do with how the program works.

The syntax of a program is never seen by the machine that executes the program. It is stripped away during the process of compilation leaving only the semantics of the program remaining. Consequently, errors which are detected during the execution of a program are semantic errors [1]. Thus: "When we cannot determine that some construct is erroneous without executing the program, we must call the error, if it then does occur, a semantic error" [ELS073, p. 265].

This notion gives rise to the following error classification scheme: "Errors detected by the compiler are syntactic while those that escape the compiler but are detected during execution are semantic" [ELS073, p. 265]. An inherent problem of this scheme, as noted by the author, is that the distinction is dependent on the intelligence that is embedded within the specific compiler for the specific language in which the program is implemented. Thus two errors which are connotatively equal could be classified as being syntactic or semantic depending on the situation in which they are detected.

From this discussion we learn that syntax and semantics, at least in terms of programming, are not absolute concepts. As a result, it perhaps is best to "live with certain language characteristics definitely semantic, others definitely syntactic, and others in between" [ELSO76, p. 265].

As for data modeling, many similarities can be drawn. First, a data model, like a program, is characterized by its syntax and its semantics. The situation here is slightly more complex, however, because a single model is a threefold entity; it consists of structures, operations, and constraints. Each of these components has an associated syntactic structure.

In terms of structures and operations, syntax has to do with the grammatical correctness of the schema or transactions as specified by the constructs of the aforementioned data definition language or data manipulation language, respectively. Borrowing the above notion of error classification as used with programming languages we can state the following: An error which is detected during the compilation of the structural schema is (probably) an error in structural syntax. Similarly, an error which is detected during the interpretation (or compilation) of a transaction is (probably) an error in operational syntax.

We have postponed our discussion of constraint syntax until now because the issue is somewhat complicated. First of all, we should consider implicit constraints and explicit constraints separately (recall that implicit constraints are inherent in the structure of s(s) while explicit constraints are specified in s(c)). Since implicit constraints constrain operations,their syntax may be considered to deal with the data manipulation language. Alternatively, they may be considered to deal with the data definition language in that they are a function of the properties of the structural schema.

We propose that if an implicit constraint issue has to do with the structural properties of the underlying DMF, then its syntactic personality is captured within the syntax of the data manipulation language. A syntactic error of this type results in the inability to specify the transaction.

If, on the other hand, an implicit constraint issue has to do with the structural properties of a specific structural schema, then its syntax (if such exists) is captured within the compiled schema. A syntactic error of this type results in the rejection or abortion of the transaction. As our primary interest lies in data

semantics vis-a-vis data syntax, the above distinction need not be pursued further.

The final syntactic issue, that of explicit constraint syntax, is dependent upon the means through which the constraint schema is specified. Most DMFs employ the data definition language in defining s(c). Some second generation DMF researchers propose a separate constraint specification language, however (see [HAMM75]).

In either case, explicit constraint syntax has to do with the grammatical correctness of the constraint schema as specified by the constructs of the language in which the schema is specified. Thus: an error which is detected during the compilation of the constraint schema is (probably) an error in explicit constraint syntax [2].

System semantics. The programming language concepts as presented thus far are of value in that they have provided us with a pragmatic (yet hardly absolute) distinction between data model syntax and data model semantics. We need simply consider that data model semantics is the totality of the model that is not syntax. Although this is the tack that is often taken when studying programming languages, we find it hardly sufficient for our purposes.

The notion that program semantics is "how a program works" does not give rise immediately to any conceptual parallels within the realm of data modeling. We will therefore focus on a less traditional treatment of semantics as provided by Edsger Dijkstra, one of the masters of programming language design [DIJK76].

The elegance of the technique lies in the fact that it deals with the semantics of a general system as opposed to program specific semantics [3]. The system can be a program, but more generally it can be a machine, a mechanism, or (presumably) a data model.

Consider a system (the design of which is goal directed) that, when started in an initial state, will eventually come to rest in a final state. If the system is viewed as being an activity, the characterization of some desired final state is called a "post (activity) condition."

In terms of inputs (arguments) and outputs (answers), we can take the position that the initial state reflects the value of the input, and the final state reflects the value of the corresponding output. Note that we assume that the final state (post-condition) depends on the initial state. Thus, in our alternative view, the answer depends on the argument that is input.

In this system the mapping from initial state space to final state space need not be functional. In other words, we allow for a number of initial states to result in the same final state (this does not imply that the system is non-deterministic as the mapping specified is from initial to final). Given the characterization of some final state, the initial states that result (upon system activation) in such a final state are said to be the states that "satisfy" the post-condition.

The characterization of all initial states that result, upon system activation, in the realization of some post-condition is called the "weakest pre-condition" for that post-condition. For our purpose, the term weakest is better understood as meaning "most general."

For a given system (S), the weakest pre-condition for the post-condition R is specified as:

$$wp(S,R)$$

Through its definition we know that any initial state that adheres to the specification of wp(S,R) will result in adherence of the final state to the specification of R whenever S is activated.

The semantics of the system are specified as follows: "We take the point of view that we know the possible performance of mechanism S sufficiently well, provided that we can derive for any post-condition R the corresponding weakest pre-condition wp(S,R), because then we have captured what the mechanism can do for us; and in the jargon the latter is called its semantics" [DIJK76, p. 17].

Since the semantics of system S are captured in the derivation of wp(S,R) from R, the semantics of S are defined by a rule which specifies how the derivation can be performed for any R. Such a rule is called the system's "predicate transformer" because when given the predicate R, it provides the corresponding wp(S,R) for S. The semantics of a system are therefore the predicate transformer of the system.

In terms of programming, the above description tells us that we thoroughly understand the semantics of a program if we know its predicate transformer. In this case the predicate transformer essentially tells us the following: for any specific result, what is the most general specification of the argument.

This concept can be expanded to include programming language semantics: "We consider the semantic characterization of a programming language given by the set of rules that associate the corresponding predicate transformer with each program written in that language" [DIJK76, p. 25].

In terms of understanding data semantics, we believe that a number of helpful concepts can be drawn from the notion of predicate transformation. First, consider that a data model (as defined through the specification of constraints, structures, and operations) is a system whose design is obviously goal oriented. Since the purpose of the system is to represent aspects of reality, activation of the system refers to the process of providing a representation, or modeling.

The system initial states, or inputs, are the real world phenomena which we attempt to represent. The system final states, or outputs, are the respective database occurrences.

Continuing along this line, we suggest that a post-condition relates to the characterization or definition of some system final state -- a given database occurrence (D). Correspondingly, a pre-condition identifies some set of system initial states which are

real world phenomena. Furthermore, for a given data model M and a post-condition D, the weakest pre-condition wp(M,D) is the most general specification of real world phenomena such that all phenomena that match the specification will be represented as D by M.

In the general description of predicate transformation we found that the semantics of a mechanism S, as captured in the derivation of wp(S,R) from R, indicates "what the mechanism can do for us." In correspondence, the semantics of the model M are captured in the derivation of wp(M,D) from D. Therefore, we propose that the semantics of the model indicate "what the model can represent;" how much of the real world meaning is included in the database representation (occurrence).

As in the general description above, a rule that specifies how the derivation of wp(M,D) can be performed for any D of M provides a definition of the semantics of M. This relates to defining the interpretation that can be awarded to any database occurrence of the model.

Finally, we propose that the semantic characterization of a DMF is given by the set of rules that associate the corresponding predicate transformer with each model that is specified using the constructs of the DMF. If the predicate transformers that are

associated by means of these rules contain a rich real world interpretation, then the DMF has an inherent semantic opulence.

A Theory of Data Semantics

At this point in the discussion we wish to direct attention towards the theoretical framework from which data modeling and data semantics concepts stem. Here, we are not concerned with "database theory" [TSIC82] which also goes by the name of "design theory" [ULLM80] and "database normalization theory" [BEER78]. Such a theory base is indeed valuable, but it is somewhat peripheral to our focus in this work.

In that database theory is founded almost entirely on the concept of functional dependency, and functional dependency is only marginally related to data semantics [CHEN76; HAMM75; ROUS75; SCHM75], the framework is not well suited to our needs. Instead, we seek a theory which deals with data models and data modeling.

Current literature indicates that a theory of this type has not yet been developed [TSIC82]. We do, however, find a specification of what the theory should consider:

"Such a theory is not only a theory of databases. It should be the beginning of a theory of data, information and knowledge as it can be operated on by computers" [TSIC82, pp. xii, xiii].

Given the realization of the fact that there is, at present, no firm theory base from which we may draw, we must suffice to piece together the theoretical concepts which are present in the relevant literature. A logical starting point is to determine where the foundations of our sphere of interest, database management, lie. "Computer science drew from mathematics and engineering. Database management draws from computer science and management studies" [TSIC82,p.xii].

We see from this diverse genealogy that database study is perhaps more eclectic than any one of its foundation areas. From mathematics, the field of database management, as any science, relies upon the fundamental laws of algebra. Beyond this it depends upon set theory, formal logic, and relation theory [BEER78; BOSA62; CHEN76; CODD70; CODD72].

The contributions from electrical engineering deal primarily with hardware concerns and the physical database [HONG81; MARC81]. Obviously, many areas are drawn from within the field of computer science. Among these are

artificial intelligence (AI) [HEND78; MINK77; MYLD75; ROUS75; WALT78], operating systems [STON81], programming languages and software engineering [BALZ79; VAND81], and data structures [CLEM81; LEHM81].

In terms of contributions made by the field of management studies, we find control theory and forecasting [BONC81], as well as a number of human factor concerns. Specifically, these are decision making [BONC81; ROUS75], learning [MINK77], perception, and interpersonal communications [CODD74; HEND78; MINK77; MYLD75; WALT78].

If we are to seek out a formal theory of data modeling, we must first provide a definition of the term "theory." While there are many possible candidates for this position, we choose to rely on the following: "A theory is a set of interrelated constructs (concepts), definitions, and propositions that presents a systematic view of phenomena by specifying relations among variables, with the purpose of explaining and predicting the phenomena" [KERL64, p. 9].

In terms of our work, the phenomena in which we are interested are the data modeling process and the semantics that are associated with such. Since we are concerned with <u>explaining</u> the phenomena as opposed to their prediction, we will attempt to "interrelate relevant

constructs in order to present a systematic view with the purpose of explaining data modeling and data semantics."

The concepts and terminology of the theory that will be presented below rely heavily upon the ideas discussed in [BILL78] and to a lesser degree those in [CHEN76], [SUND74], and [TSIC82]. When theoretical constructs relate to the work of others we will indicate this with appropriate referencing.

We will see that the theory proposes that various types of semantics are given by the mapping of facts from one representational form to another. Since semantics have to do with meaning, a pivotal element in the theory is that there exists a representation of facts from which a common understanding can be derived. There is required some means by which a fact can be "anchored" to its meaning; a meaning which is universally understood.

What we seek, then, is some medium of discourse through which facts can be represented in a form that provides a consonance of comprehension. Such a medium is natural language.

The natural language representation of an elementary fact is in terms of an object reference, a property or relationship reference, and a time reference [TSIC82]. An object can be concrete or abstract; it can be atomic or

compound. It is anything about which assertions can be made [BILL78].

Properties and relationships are asssertions about objects. In a natural language representaion, a relationship is denoted by a verb (e.g. "is employed by"). The natural language denotation of a property is by means of a predicative expression [BILL78] (e.g. "to be a student").

For the sake of simplicity, we will avoid using the temporal reference within the specificaion of a fact. It is this statement of time which allows the dynamic nature of reality to be captured. By fixing time as a specific instance, say t, we can conceptualize the factual representaion of reality as it exists at that time instance; we have a "snapshot" [HAMM75] of the real world. Unless otherwise specified, then, every representation referred to in the following discussion is a "time t" representation.

We begin the discussion by introducing the concept of <u>reality</u> -- the totality of real things. As a surrogate of reality we will use the totality of all possible facts which can be stated about reality.

For any database application, only a portion of reality is of interest. Consequently, we need be concerned with only a subset of all possible facts that can be stated about reality. The chunk of reality which is of concern to the application will be referred to as the universe of discourse [BILL78; BONC81; ROUS75; WONG77]. Our factual representation of the universe of discourse, the "slice of reality" [SUND74], will be called the real world state [BILL78].

Extraction of the universe of discourse (and corresponding real world state) from the total reality is achieved through application of the processes of selection and perception. The existence and relevance of an object is determined by our percepts: "Objects are born when people become interested in them. They die when people stop being interested in them. They change when they are sufficiently different that people perceive them as different objects" [TSIC82].

The real world state therefore comprises a set of facts that exist as assertions about the objects which we are both interested in and aware of at time t. If we consider for a moment the dynamic nature of the real world we can define the set of imaginable real world states. This is the set of all real world states which may

(according to the users) exist at some point in time (see figure 3.1).

Note that we have not yet imposed a natural language structure upon the facts; they are conceptualizations only. Before the facts can be structured, we must consider the possibility of "conceptually different views" [BILL78].

Because our conceptualizations depend on our individual percepts, it can be imagined that two different users could have views of the universe of discourse which are conceptually different. As a result of such a situation, the nature of the facts and assertions that constitute the real world state would differ from one user to another.

A commonly used example has to do with the various ways in which "color" can be regarded. If color is thought of as being an object (e.g. "red"), then assertions about colors can be made. Such assertions can deal with properties of the color ("has wavelength"), or they can specify relationships among colors and other objects ("has color"). On the other hand, color can be regarded as being a property which is used in making assertions about other objects ("to be red"). In this case, further facts about colors cannot be specified.

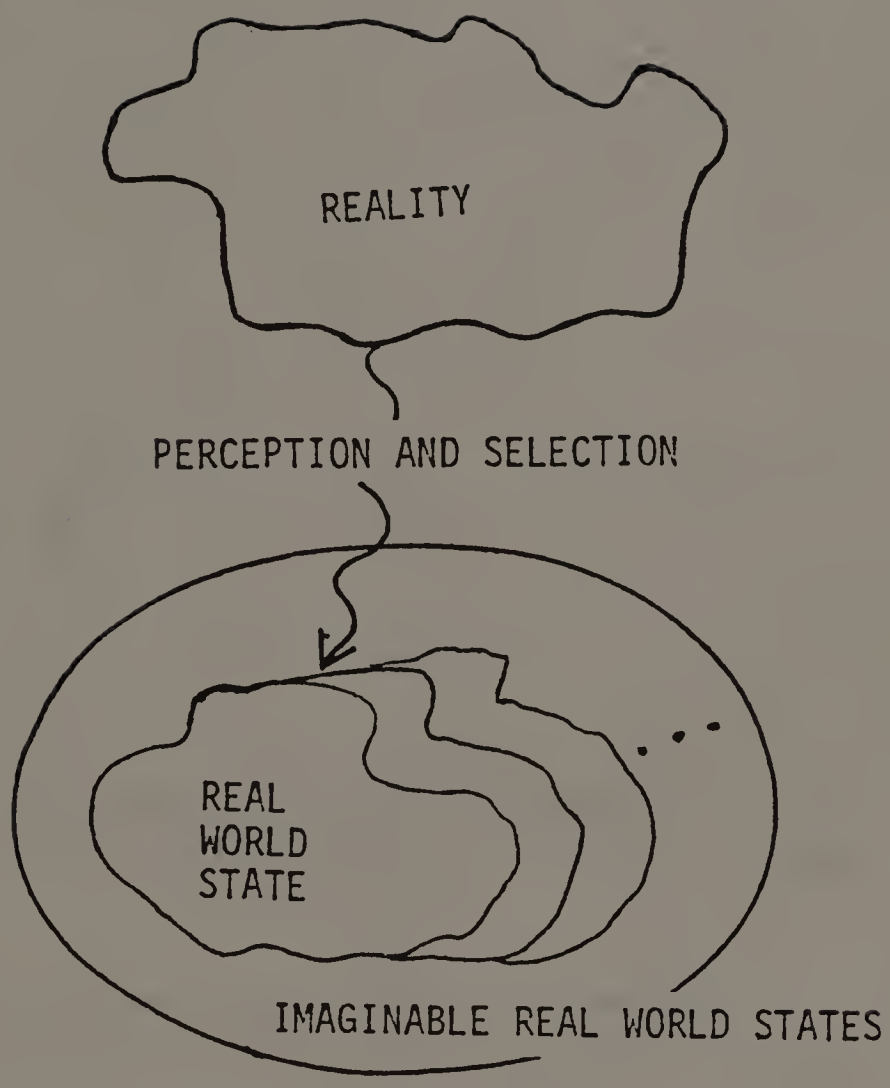

REALITY
PERCEPTION AND SELECTION
REAL
WORLD
STATE
IMAGINABLE REAL WORLD STATES

FIGURE 3.1

Because the process of perception cannot be formalized, there is no way of proving the absence or presence of conceptual inhomogneity. If such differences exist in our real world state, all subsequent representations of the state will result in misunderstanding on the part of the users. If we are to continue, then, we must assume that the real world state is vacuous of conceptually different views.

An analagous assumption which will be made is that of factual consistency -- a fact and its negation will not be contained in a given real world state. Furthermore, if a fact is not asserted, it is false [4]. This is known as the "closed world assumption" (see [REIT78]).

A natural language specification of the real world state will be called the state description. This state description comprises the set of elementary predicative sentences which describe the real world state completely [BILL78]. An elementary sentence is one which cannot be specified as a conjunction ("and") of other sentences; it is atomic. A sentence in the state description either states that a relationship among a number of objects holds or that an object possesses a property.

Each elementary sentence is made up of three distinct components: a subject, a predicate, and an object-part [5]. The subject and object components are denotations that need not be simple. The object-part may denote any number (including zero) of objects. Each of these objects, as noted earlier, may be compound. The subject denotation can also be structured.

Since the state description is a natural language description of the real world state, the semantics of natural language are given by the mapping of elementary sentences to real world state facts (see figure 3.2). Note that we assume that all users agree on the overall truth of the state description. Thus, there is a consensus that each elementary sentence represents a fact that is true with respect to the real world state. This assumption presupposes the existence of a common understanding of the natural language that is used to represent the real world state.

The discussion thus far has dealt with what is commonly referred to as the infological realm of data modeling. The infological realm is concerned with mapping the real world into basic human concepts [TSIC82]. Since our ultimate goal involves the understanding of a machine implemented representation, we must consider the

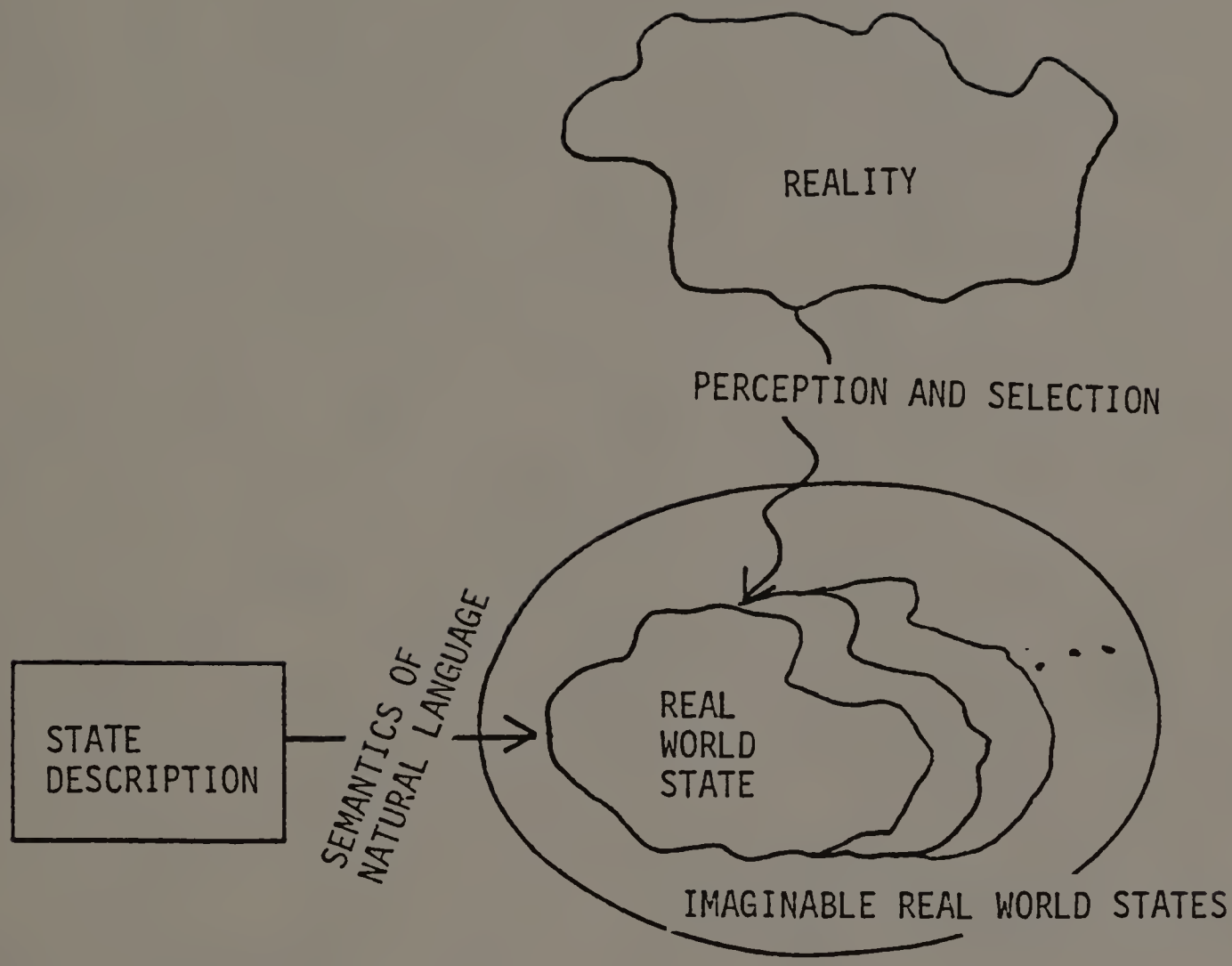

REALITY
PERCEPTION AND SELECTION
STATE DESCRIPTION
SEMANTICS OF NATURAL LANGUAGE
REAL WORLD STATE
IMAGINABLE REAL WORLD STATES

FIGURE 3.2

datalogical realm of data modeling. Here, we map from the infological representation (state description) to a machine representation.

Neither the real world state, nor its infological representation, the state description, lends itself to a formal treatment. The nature and number of constructs are overwhelming. By abstracting the set of imaginable real world states we can specify the abstract world model. The abstract world model is a set of abstract states each of which corresponds to a real world state.

An abstract state, as described in [BILL78], is a mathematical structure which is imposed upon a set of entities (E), a family (from E) of mathematical relations (R), and a family (from E) of types (T) [6]. The structure is as follows:

1. Every type (T(i)) is a subset of the set of entities

$$(T(i) \varepsilon T) \subseteq E; \forall i$$

2. The set of entities comprises the elements of the family of types [7]

$$E = \{T(1) \cup T(2) \cup \ldots T(n) | T(i) \varepsilon T\}; \forall i$$

3. Every relation is a subset of the set of entities and has an associated rank N(i)

$$(R(i)^{N(i)} \varepsilon R) \subseteq E; \forall i$$

4. An entity may belong to two different types [8]

$$T(i) \cap T(j) \neq \emptyset; i \neq j$$

All abstract states of the abstract model are constrained to have the same number of types and relations. Notice that there is no constraint on entities across states. Also, type and relation sets need not contain any elements.

$$T(i), R(i) = \emptyset$$

The abstract state is related to the real world (state) by means of the standard interpretation. The standard interpretation maps phrases in the state description to components of the abstract state. Furthermore, it maps sentences in the state description to the value true or false.

Noun phrases in the state description are mapped to entities of the abstract state such that only those noun phrases which denote the same real world object are mapped to the same entity. Verb phrases are mapped into relations or types depending on whether they denote, respectively, relationships or properties.

Sentences of the state description are mapped to the value true if the mapping of its constituent phrases is consistent with reality. In the situation where the sentence states that an object possesses a property, it is mapped to the value true iff (if and only if) the type into which the verb phrase is mapped contains, as an element, the entity to which the respective noun phrase is mapped.

In the case where the sentence denotes the existence of a relationship among a set of objects, it is mapped to the value true iff the entities to which the subject and object-part of the sentence are mapped form a tuple which appear as an element of the relation into which the verb phrase is mapped.

If each and every sentence in the state description is mapped to the value true, the respective abstract state is said to be a "model" (as used in formal logic) of the state description. If we assume that the abstract state is a model of the state description, and we know (by prior assumption) that the state description is true with respect to the real world state, then the abstract state can be referred to as an abstraction of the real world state [BILL78] (see figure 3.3) [9].

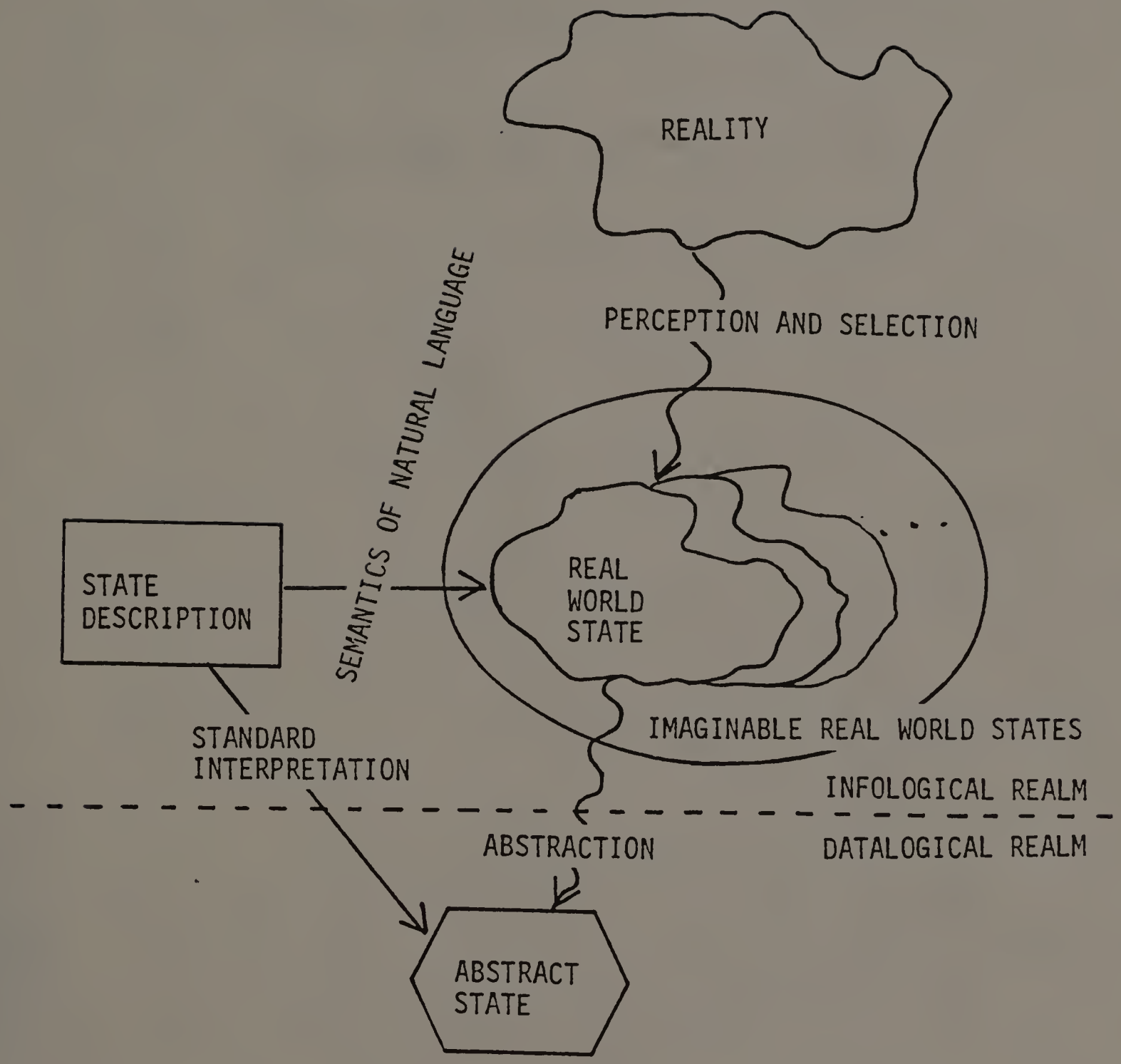

REALITY
PERCEPTION AND SELECTION
SEMANTICS OF NATURAL LANGUAGE
STATE DESCRIPTION
REAL WORLD STATE
IMAGINABLE REAL WORLD STATES
STANDARD INTERPRETATION
INFOLOGICAL REALM
ABSTRACTION
DATALOGICAL REALM
ABSTRACT STATE

FIGURE 3.3

Since the abstract state is a syllogistic abstraction of the real world state, the components of the abstract state; entities, types, relations; are abstractions of their respective real world counterparts: objects properties, and relationships. For example, the type T(p) (of the real world state) is an abstraction of the property p (of the abstract state) because the verb phrase in the state description which denotes the property p is mapped into T(p).

The real world state and the abstract state, as described here, hold a close correspondence to the top two levels of the "multilevel views of data" scheme as detailed in [CHEN76]. According to Chen the Level 1 view, which is called "information" is concerned with entities and relationships which "exist in our minds." Because these entities and relationships carry the qualification that they are to be eventually represented in the database; Level 1 corresponds to the real world state.

Level 2, "information structure," provides a data grouping and value representation of Level 1 according to Chen. This representation describes formally the objects, properties, and relationships as does the abstract state. Consequently, our theoretical construct of abstraction relates to the notion of Level 1 --> Level 2 traversal in Chen's multilevel structure.

At any point in time the set of data instances that are contained in the database is the database occurrence at that time. The syntactic correctness of the database occurrence is determined by the database schema. The schema is the result of the application of a set of generating rules which consist of a structure specification as defined by a particular data modeling formalism [TSIC82].

A data instance can be regarded as the representation of one or more facts. It is a sentence that is specified in a formal language. The grammar of this language is provided by the database schema [BILL78].

At each instance of time the set of facts which is represented by the database occurrence corresponds to the abstract state for that time. The schema, on the other hand, is static so it corresponds to the set of all abstract states -- the abstract world model. Given this realization we are provided with a specification of the desired semantic constructs: The semantics of the database (occurrence) are given by the mapping from the database to the appropriate abstract state. The semantics of the database schema are found in the mapping from the schema to the abstract world model [BILL74] (see figure 3.4).

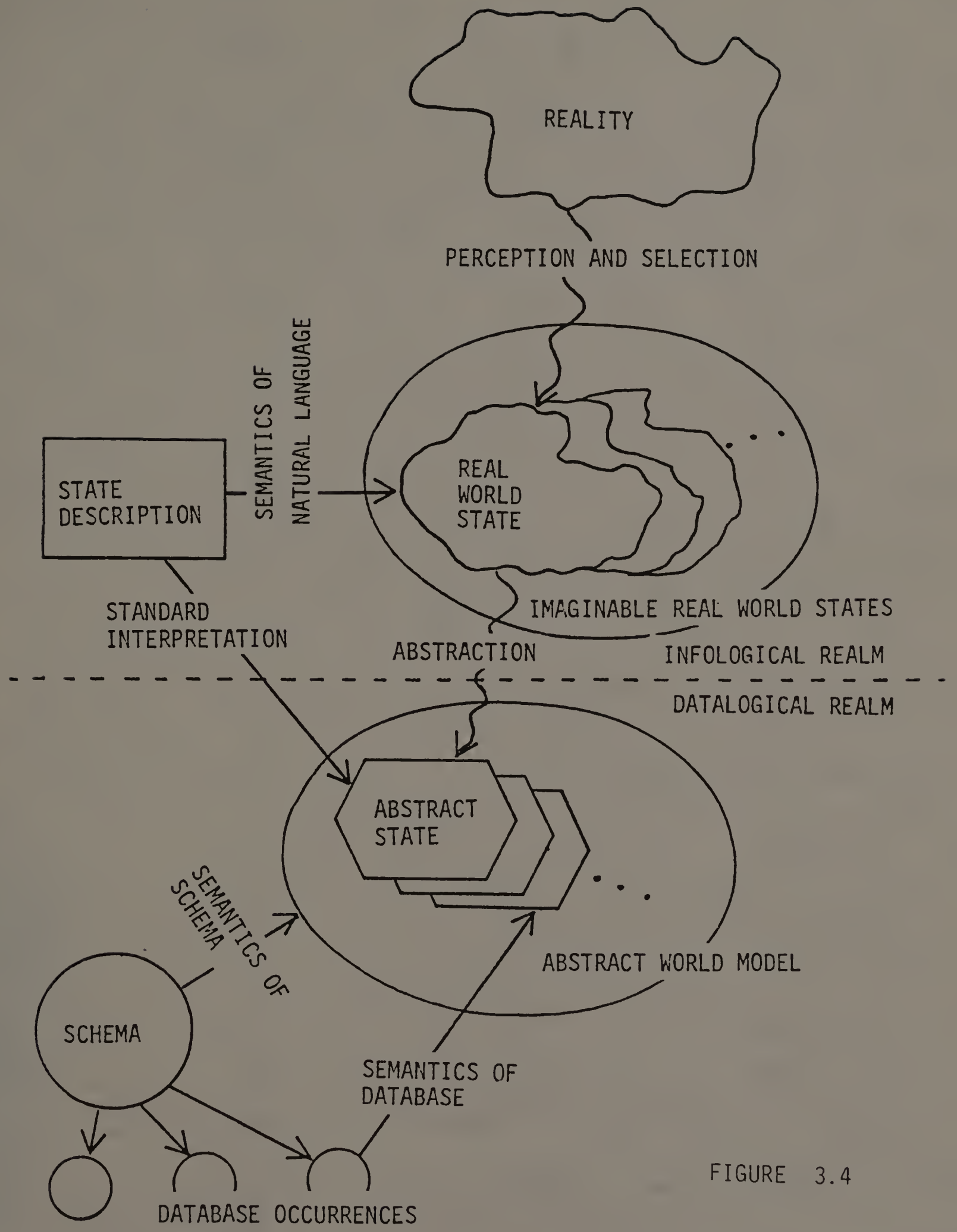

REALITY
PERCEPTION AND SELECTION
SEMANTICS OF NATURAL LANGUAGE
STATE DESCRIPTION
REAL WORLD STATE
STANDARD INTERPRETATION
IMAGINABLE REAL WORLD STATES
ABSTRACTION
INFOLOGICAL REALM
DATALOGICAL REALM
ABSTRACT STATE
SEMANTICS OF SCHEMA
ABSTRACT WORLD MODEL
SCHEMA
SEMANTICS OF DATABASE
DATABASE OCCURRENCES

FIGURE 3.4

In regards to our predicate transform treatment of data models, the former construct, the semantics of the database occurrence, relates to the derivation of the respective weakest pre-condition (which is depicted by the abstract state) from the post-condition (database occurrence). The latter construct, the semantics of the schema, relate to the predicate transformer of the model. Therefore, "semantics of the schema" as used here is the "semantics of the mechanism" (model) as used previously.

Data Semantics in Database Literature

Database literature supplies us with a number of definitions of data semantics. Among the less intricate of these is the following: "By the semantics of data we mean the 'meaning' of the data" [KERS76, p. 58]. In that we have provided an extensive theoretical backdrop for the notion, we feel that this more pragmatic description is quite acceptable. We will therefore adopt it as our working definition of semantics from this point forward.

Before we pursue our study of specific modeling concepts and individual DMFs, we will briefly mention a number of data semantic related issues that appear in the

relevant literature. First, we find a description of the function of semantics within our sphere of interest: "The role of database semantics is to ensure that stored information accurately represents the enterprise" [BEER78, p. 113]. This statement serves to clarify and restate the ultimate goal of our pursuit.

Some authors cite the area of linguistics as a possible source from which a further understanding of semantics can be drawn. In [KERS76] for example, we find reference to the possible use of linguistic "predication analysis" as a means to a better understanding of data model semantics and related syntax. It is suggested that such a methodology would allow for the categorization of data models within a "semantic spectrum." Placement of a model within this spectrum, which ranges from "surface semantics" to "deep semantics," would be a function of the level of abstraction that is embodied in the information structures of the model.

Also from the field of linguistics, we find reference to the "pragmatics" of a statement vis. its semantics [LEE76]. Given a syntactically correct statement in some (natural) language, the semantics is the meaning that is intended by the originator of the message. This may differ from the meaning that is interpreted by the receiver: the pragmatics of the message.

Reference to linguistic pragmatics, as cited here, is made in terms of programming applications, but we see the possibility of a conceptual parallel in terms of data modeling. The interpretation of the database occurrence that is made by the user is the "pragmatics" of that database occurrence. This may differ from the semantics of said database occurrence. In the event of a discrepancy, we see the difference between the interpretations as encapsulating the "semantic ambiguity" of the database occurrence (and possibly the ambiguity inherent in the underlying model or DMF).

In [CODD79] we observe the distinction between "atomic semantics" and "molecular semantics." Atomic semantics deals with the search for meaningful (information) units that are as small as possible. Molecular semantics, on the other hand, has to do with the search for meaningful units that are larger than relations [10]. The former notion relates to the concept of defining modeling construct primitives. The latter relates to the ubiquitous concepts of aggregation and generalization [SMIT77a; SMIT77b]. These three concepts; construct primitives, aggregation, generalization; will be discussed in detail later.

Finally, from the same source, we discover a warning as to the possibility of frustration in our pursuit of data semantics: "The task of capturing the meaning of data is a never ending one. So the label 'semantic' must not be interpreted in any absolute sense" [CODD79, p. 398].

FOOTNOTES

[1] Obviously, not every semantic error is captured during program execution. We can conceive of a semantically erroneous program as achieving "normal" termination. In this case the results are faulty, but the faults are likely to go undetected.

[2] Note that s(c) and s(s) are not necessarily separate entities. In the case where s(c) is specified in terms of the DDL they, in all probability, are not distinct entities. When s(c) is specified in terms of a CSL they are distinct.

[3] The methodology is applicable, with slight modification, to non-deterministic systems as well as to deterministic systems. This distinction need not concern us here.

[4] Some infological models utilize the notion that a fact that is not true may be "close" to being true and should therefore not be regarded as being false. Representation of this concept is achieved by allowing for the specification of an "infological distance" between messages.

[5] A predicate which is followed by one or more object terms can serve as a formal representation for an elementary sentence. Such a notation allows us to avoid some of the ambiguity that is inherent in natural language. This obviously is the basis of predicate logic and predicate calculus.

[6] Consider a non-empty set:

$$s = \{a,b,c\}$$

A "sigma field" (s) of S is the set of all possible subsets of S. Thus:

$$s = \{\emptyset,(a),(b),(c),(a,b),(a,c),(b,c),(a,b,c)\}$$

A family (F) from S is a subset (not necessarily proper) of s

$$F \subset s$$

Thus, F is a set of subsets of S.

[7] Most DMFs include such a constraint. In this case the model is said to be "strictly typed." Strictly typed models are discussed in a later section.

[8] Statements 2 and 4 imply that T is a covering of E.

[9] This application of the term "abstraction" differs slightly from its common usage in data modeling. Abstraction as generally used in data modeling refers to the process through which the commonalities of objects are emphasized by means of ignoring detail and individual difference. This is discussed in detail in a later section.

[10] A proper generic interpretation of this sentence can be gained by replacing the term "relation" with "database entity or database association."

C H A P T E R I V

OBJECTS AND OBJECT GROUPS

In this chapter we discuss the basic objects of databases and data models. In addition, we consider how these basic objects are collected into groups.

Database Objects and Primitives

In reviewing the data modeling theoretical framework that was presented above, it becomes obvious that there are four general kinds of objects in which a data model is interested. Specifically, these are entities, values, associations among entities, and properties of objects. Interest in these types of objects is characteristic of all data modeling formalisms (DMFs). This includes first generation as well as second generation formalisms. In studying the specifics of various DMFs we will find that the treatment of these objects is quite diversified, however.

Database objects. Entities go by a number of different names such as "concept" [ROUS75; SU79], "thing" [SENK75], and "independent object" [SCHM75]. These represent the real world constructs whose behavior we wish to emulate. While their incarnate counterparts may not be simple, entities can be regarded as being indivisible for modeling purposes [SMIT77a].

An entity can be used to represent a real world concrete object or, alternatively, it can depict an event (it has been noted that making the distinction between such can hamper the flexibility of the model [HAMM81]). The RM/T data model [CODD79] distinguishes three different types of entity: characteristic, associative, and kernel. Each of these types maintains its own generic classification scheme.

Entities are the main focus of interest in a data model. Concern for the other database objects is due to the fact that they ultimately describe entities [KERS76]. Thus we consider associations only because they relate entities. Similarly, we deal with values because they provide meaning to the properties that are used to describe the entities.

Depending on the DMF, associations can have properties or not. The well known Entity-Relationship Model [CHEN76] is of the former category. In the RM/T model [CODD79], both situations are allowed. If the association is awarded the status of an entity, then it may take on properties. Otherwise, no properties are allowed. In the former case the object is called an "associative entity;" the latter object is a "nonentity association."

In some DMFs entities which are used to describe other objects can themselves be described. This is true of the "characteristic molecule type" of RM/T [CODD79]. Also, the Basic Semantic Data Model [SCHM75] can be viewed in this way. Here, "characteristic objects" are those which exist only to describe "independent objects" (entities) and other characteristic objects of the world.

This phenomenon can be looked at in two different lights as noted in [KERS76]: (1) characteristic objects are the properties and independent objects are the entities, or (2) there are no properties but there are two classes of entity -- independent and characteristic. Note that the first view allows for nested properties. Some semantic network based models utilize the concept of nested properties in a similar way (e.g. [ROUS75]).

Database primitives. The fundamental data model objects; entities, associations, properties, and values; can be thought of as being the construct primitives of data modeling. Here, the term "primitive" has a more comprehensive connotation than it holds in its traditional programming language usage. We use the term within a data modeling context to refer to an object which the user considers to be atomic. In other words, from the view of the user, there is no need to further dissect the object [1].

We believe that the aforementioned constructs meet this specification. For example, there is no reason to conceptualize the notion of "part of an entity" or a portion of an association. True, these objects are further described in terms of their properties, but they are not further partitioned.

We will also use the term "primitive" in its more traditional sense; as it is used in computer programming. Briefly, programming distinguishes two types of primitives: data primitives and programming primitives. We will see that both of these have analogous concepts which are of value in our discussion of data modeling.

Data primitives stipulate the most basic data forms (types) that are available for use by the application at hand. In terms of computer programming, common data primitives are integer, character, Boolean, etc. [2]. From the view of the programmer these can be assumed to be provided by the hardware. In terms of data modeling, the concept of a data primitive is exactly the same. Assignment of a data value to a primitive data type is used to aid in the maintenance of basic data integrity.

In computer programming, the programming primitives are the most basic operations that are available for use by the application. High level language primitives are of the form READ, CALL, etc. [3]. In data modeling we use a concept that is much the same. The set of operations (the set "O" in our data model notation of chapter 2) comprises the primitives of the respective DMF. In order to avoid the ambiguity that would stem from use of the term "programming," we will refer to this set as the operation primitives of the DMF.

Data Typing

In the above text we set forth the proposition that entities and associations be considered modeling primitives. The rational provided was that the user is not concerned with a further dissection of these objects. The opposite process, grouping these objects into categories is of great value, however.

In natural language discourse we often make reference to a "higher level" object which constitutes a set of "lower level" (more primitive) objects. For example, in the statement "Doctoral students make good lovers," the term "doctoral students" denotes all people who qualify for consideration as a doctoral student. In other words, reference to the single term doctoral students implies reference to the entire set.

In data modeling we use a similar concept which is called "classification," or "typing." A *type* is an object which is formed by the grouping of homogeneous objects. The individual objects that form a type are called "instances," "occurrences," or "tokens" of the type. The process of decomposing a type into its primitive token instances is called "instantiation" (note that this is the inverse of classification). In general, data models

classify both entity and association tokens into types. The schema can be considered as a collection of these types [TSIC82].

The type-token concept roughly corresponds to the set theoretic notion of intension-extension. The intension of a set is a description of the constitution of the set; it is definitional. Set intension is often specified in terms of a membership rule or predicate which defines which occurrences are permitted.

The extension of a set is a specification of the actual set occurrence; it is representational. For example, given the set intension:

$$\{x | x = \text{positive even integer, and } 2 \leq x \leq 8\}$$

a respective extension is:

$$\{2,4,6,8\}$$

In terms of data modeling, a type can be considered as an intension in which case the tokens of the type are an extension. Since a schema is a collection of types, it can be considered as the intension of the database. Each corresponding database occurrence is therefore an extension of the intensional schema.

Strict vs. loose typing. A DMF can be either strictly typed or loosely typed. In a strictly typed model each object must, at all times, be a member of some type. If an object does not fit naturally into a type there are two alternatives: (1) force membership of the object into some type or (2) remove the object from the domain of the model. In a loosely typed model uncategorized objects are allowed to exist because typing is optional.

All DMFs that deal with the datalogical realm of data modeling are of the strictly typed variety [TSIC82]. Some infological models are loosely typed (see[SUND74]) [4]. Since our work centers around the datalogical sphere, all DMFs in which we are interested are strictly typed.

Strict typing results in two basic detriments as noted in [TSIC82]:

1. The DMFs have an inherent inflexibility in representing subtle differences -- they force homogeneity

2. The groupings must (in most cases) be specified a priori -- types can not evolve dynamically

There are, however, a number of distinct advantages to be gained by using strict typing. The major advantage is that typing reduces the number of objects that need to be dealt with. In terms of conceptualizing the ways in which objects are related to each other, tokens need not be considered. All inter-object concerns can be faced in terms of less cluttered inter-type issues.

A second advantage relates to the properties of typed objects. Since a type is defined in terms of token object homogeneity, the homogeneous properties of all tokens of a type can be abstracted to the type level. In other words, the properties can be considered to "belong" to a single higher level object rather than to a multitude of primitive objects. As this issue is fundamental to the study of second generation DMFs, it will be discussed in great detail later.

Typing in AI. The less traditional artificial intelligence (AI) approach to data modeling enforces (strict) typing of objects, but the methodology differs, in some cases, from the standard (non-AI) database management (DM) modeling approach. The terminology may differ in that some AI based models refer to tokens and types as "concepts" and "classes," respectively. The connotation of these objects remain the same, however.

A more extreme divergence lies in the fact that some AI models mix tokens and types in the same structure as do infological models (see [SUND74]). Unlike the infological models, however, the AI structures maintain a strict distinction between the generic and specific objects (note that the AI models are strictly typed).

As an example, consider that the semantic net(work) of an AI model is the rough equivalent of the schema of a standard DM data model. While the DM schema provides a database intension only, the semantic net provides both database intension and database extension. Although these types and tokens are combined within a single network structure, a distinction between the categories is maintained. All type (class) objects exist in the "upstairs" of the net while all tokens (concepts) are "downstairs" objects (see [ROUS75]).

The role approach. In recent years a number of modifications of the standard data typing concept have been proposed. Among these suggestions is the role model [BACH77]. In this approach an entity is characterized by the roles it plays [5].

The typing of objects is twofold in the role model. Categorization is by role type (roles with similar properties) and by entity type (entities with similar properties). Role types do not overlap nor do entity types. Furthermore, roles and entities are not subsets of each other.

Relationships are among roles rather than among entities. Entities are mapped to roles, so entity objects are ultimately associated in terms of the roles they play. The cardinalities of the mathematical mapping between role and entity types is allowed to be complex (many:many). Thus one entity token is allowed to play a number of different roles and each role can be played by many entities.

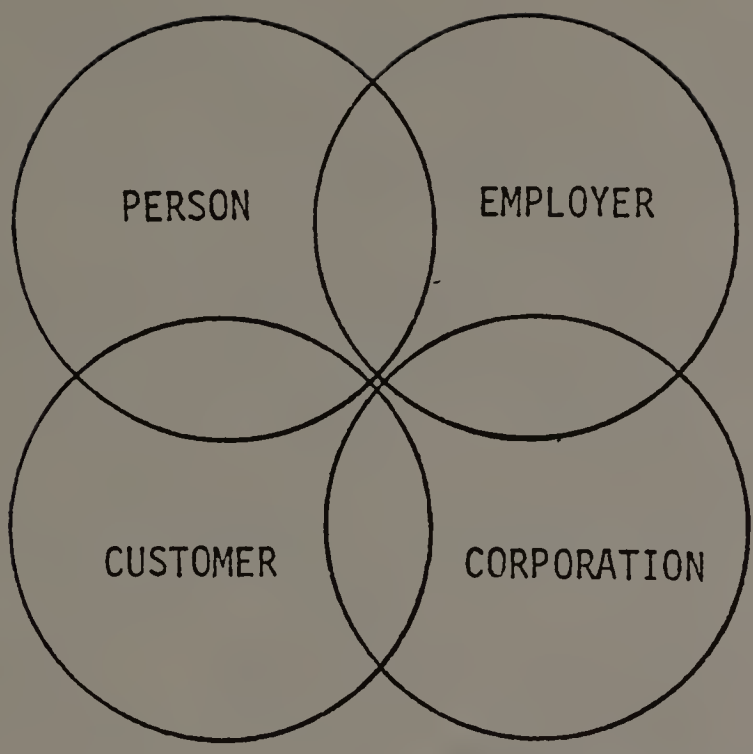
Because the constraints implied by the above descriptions are lax, some explicit constraints can be identified. Specifically:

1. A role can be declared as being shareable or non-shareable depending on whether it may be associated with more than one entity

2. A role-entity association can be declared as being essential or nonessential depending on whether an entity occurrence implies the existence of a role occurrence

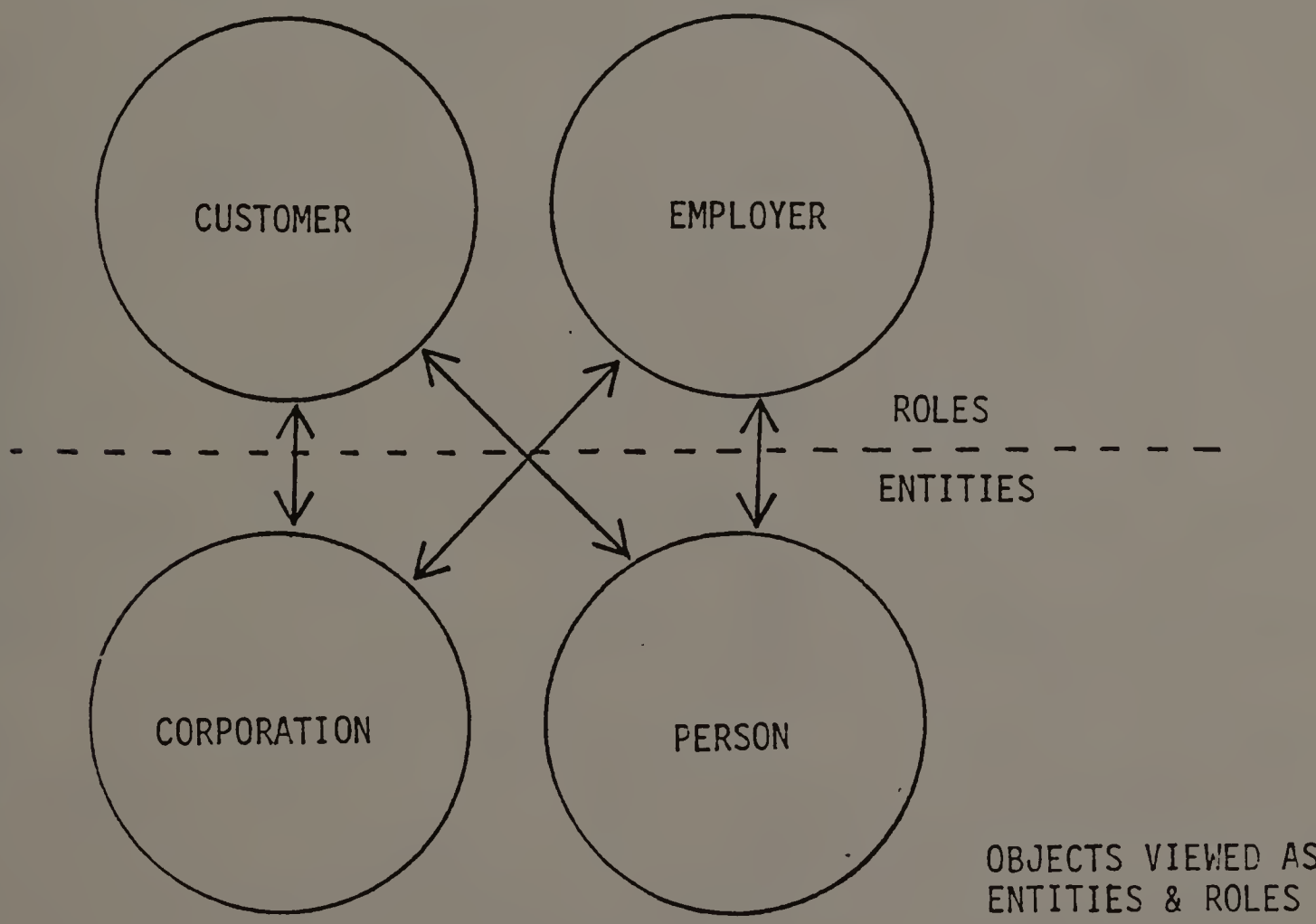
The ultimate purpose of the role concept is to provide clarification in the situation where traditional entity types overlap. Consider, for example, the object types "person," "employer," "customer" and "corporation;" all of which are of concern to the enterprise. If all such objects are considered entities, as is the usual procedure, then each of these object categories is an entity type.

Under this approach a conceptual problem arises when we consider the overlap of types. A customer may be a person or a corporation. At the instance level, then, tokens are not homogeneous within a type; they are very different [6]. This is illustrated in figure 4.1a.



OBJECTS VIEWED AS
ENTITY TYPES

FIGURE 4.1a



OBJECTS VIEWED AS
ENTITIES & ROLES

FIGURE 4.1b

The role model methodology alleviates this anomaly by considering that "customer" and "employer" are not entities. Rather, they are roles that are played by entities. Thus, a corporation entity token which is represented in the database can play the customer role or, alternatively, it can play the employer role. Likewise, a given person occurrence can be a customer or an employer in the role that it plays within the enterprise (see figure 4.1b).

In the next chapter we consider how the concept of object grouping can be extended so that types are themselves collected into groups. The process is called "abstraction."

FOOTNOTES

[1] As noted earlier, this is akin to the notion of "atomic semantics" [CODD79].

[2] Obviously, the level of abstraction of the data primitives is dependent on the type of application. In assembly programming, for example, data primitives are in terms of bytes or even bits.

[3] As with data primitives, consideration of an operation as being a programming primitive is application dependent. A programming primitive of a high level language is viewed as a macro(instruction) in a low level language. For instance, the COBOL ADD primitive corresponds to an assembly language macro comprising the LOAD, ADD and STORE primitives.

[4] Recall that the infological realm deals with describing the real world in terms of basic human concepts. As such, it ignores the attributes of the storage hardware and can consequently allow great flexibility.

[5] A number of DMFs incorporate the notion of "roles" into their framework. Among these are the relational model (see [CODD79]), and the E-R model (see [CHEN76]). The concept is of secondary importance in these models, however.

[6] In a record based model this phenomenon is called "horizontal inhomogeneity" [KENT79].

C H A P T E R V

DATA ABSTRACTION

In the previous chapter we described how token "primitive" objects can be gathered together to form a more generic object type. This grouping process was called categorization. The reverse process, decomposing a type into its token objects was called instantiation. A section was devoted to this concept of typing because it is fundamental to all data modeling formalisms (DMFs) in which we are interested; all datalogical data models are strictly typed [TSIC82].

In this chapter we will learn that categorization is a special case of a very general data handling construct: data abstraction. Abstraction, as used in second generation data modeling, is defined as follows: "An abstraction of some system is a model of that system in which certain details are deliberately omitted. The choice of the details to omit is made by considering both the intended application of the abstraction and also its users. The objective is to allow users to heed details of the system which are relevant to the application and to ignore other details" [SMIT77b, p. 105].

If a data model is to be of any value it must be abstract. What is gained through the use of abstraction is as follows: it allows to filter through only those portions of the database with which we are concerned. Put in alternative terms, abstraction allows us to "see the forest (information content of the data) as opposed to the trees (individual values of the data)" [TSIC82, p. 5].

As noted above, abstraction deals with ignorance of details about some object. When this detail is ignored the object is said to be "more abstract." In general, abstraction involves the representation of a number of less abstract objects as a single more abstract, or "higher level" object. The inverse of the abstraction process is called "decomposition." Thus, decomposition involves breaking a single more abstract object into a number of less abstract component objects [1].

Database management systems (DBMSs) distinguish two forms of abstraction: real world abstraction, and implementation abstraction [SMIT77a]. Real world abstraction allows the user to momentarily ignore details while conceptualizing some aspect of the world. This type of abstraction is therefore temporary.

Implementation abstraction, on the other hand, is permanent. This is used to continuously hide details of the structural implementation of the database (e.g. we, as database users, need never think of the complex pointer system that is used to implement a simple database relation) [2]. Since our interests lie in the realm of conceptual modeling vis. DBMS implementation, we are more concerned with abstraction of the real world variety. The term "abstraction" will hereunder refer to real world abstraction unless stated otherwise.

Because objects can be classified as being more abstract or less abstract, we can speak of abstraction as being "top down" or "bottom up" (note that top down abstraction is decomposition). Both of these approaches are useful in data modeling. Bottom up abstraction enables us to understand a complex phenomena. Top down abstraction allows us to design a complex object [TSIC82].

If a data model is to use abstraction to its full (semantic) potential, names given to abstract objects should be restricted to be natural language nouns. This allows for true abstraction to be distinguished from "spurious groupings of distinct concepts" [SMIT77a, p. 107]. Furthermore, each abstract object should be uniformly accessible independent of its level of

abstraction; there should be a one-to-one correspondence between objects and their respective representations in the database [SMIT77a] [3].

Since the simultaneous advent of two now classic articles, [SMIT77a] and [SMIT77b], real world abstraction has been discussed in terms of two distinct categories: aggregation abstraction, and generalization abstraction. The former has to do with the grouping of heterogeneous objects to form a more abstract representation. The latter refers to the grouping of homogeneous objects to form a more abstract representation.

Although the concepts of aggregation and generalization abstraction are fundamental to second generation DMFs, they are by no means new ideas. Each has a discernible heritage. Aggregation abstraction, as we now know it, primarily developed out of early software and (first generation) database research [4]. Our present understanding of generalization, on the other hand, has stemmed from artificial intelligence (AI) research [SMIT77b; WONG77].

Aggregation Abstraction

Aggregation, called "cartesian aggregation" in the RM/T data model [CODD79], is an abstraction through which an object gains its structure by combining a set of constituent objects. In terms of our aforementioned data modeling construct primitives, aggregation refers to the transformation of an association among a number of named (heterogeneous) entities into a single higher level named entity.

The reverse of aggregation is called "aggregate decomposition." In the domain of programming, aggregate decomposition corresponds to the process of stepwise refinement as used in program design (see [DIJK72]).

Aggregation abstraction is a "natural" modeling concept because it is used often in natural language expression of relationships. Consider, for example, the following relationship:

> "Pupil P received a grade G in a class of the course numbered C-NO with C-H credit hours and description D taught by instructor I during semester S in room R" (from [SMIT77a, p. 406]).

This phrase; which defines an 8-ary relationship among the entities P, G, C-NO, C-H, D, I, S, R; contains two inherent aggregation abstractions and can be defined as a single aggregation abstraction. The term "course" is an abstraction of the primitive level objects course number, credit hours, and description. The term "class" is an abstraction of the abstract object course, and the primitive objects semester, instructor, and room.

Furthermore, the abstract object "class," coupled with the primitive objects pupil and grade can be further abstracted to form a higher object, say "enrollment." This is depicted in figure 5.1. Note that abstraction through aggregation is inherent in first generation DMFs in that each record (relation) depicts the abstracted aggregate of its component objects.

The above example makes clear the fact that abstraction through aggregation is hierarchical in nature. At each level of the hierarchy, the component objects are attributes of the more abstract object that is formed by their aggregation.

The artificial intelligence (AI) community represents aggregation abstraction in terms of the "PART-OF" relationship. Here, a PART-OF relationship links each component part to the higher level abstracted

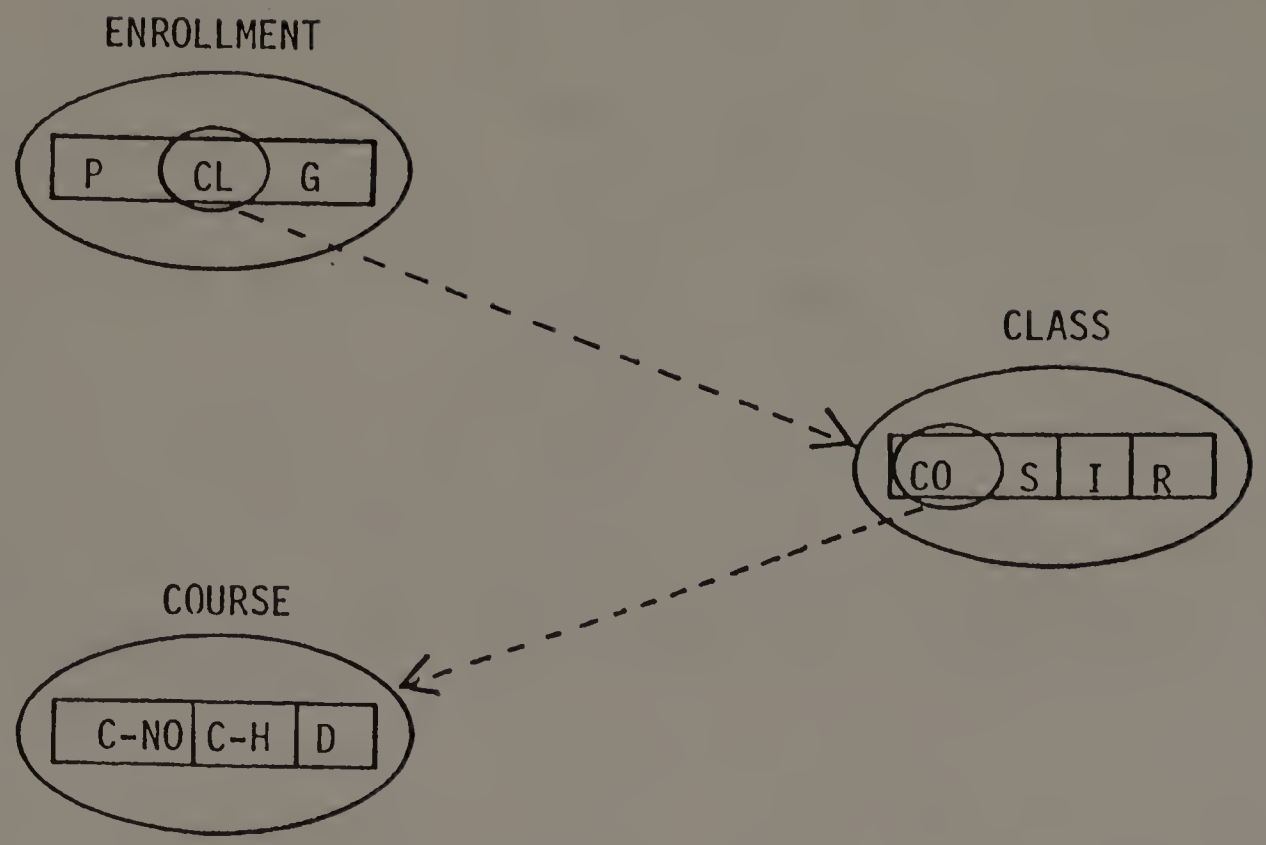


FIGURE 5.1

AGGREGATION
ABSTRACTION

object. In regards to syntactic issues, this approach is quite different from the traditional (non AI) data management methodology. Explicit relationships are defined between the object and its components. The semantics of the aggregation hierarchy, taken as a whole, are identical, however. The AI version of figure 5.1 is shown in figure 5.2.

Some second generation DMFs define a variation of the aggregation abstraction concept to account for the situation where there are no well defined attributes of the lower level object instances to specify their grouping in the formation of higher level object(s). In this case token objects of a number of heterogeneous types can enter and leave their affiliation with one another where the affiliation itself represents an entity. Consider the semantics of membership in professional organizations, for example. The heterogeneous members which dynamically make up an organization define our interest in the organization.

In such an abstraction, the set of types from which tokens may be combined provide a "covering" of the abstract object [5]. This notion is called "cover aggregation" (vis. cartesian aggregation) in the RM/T data model [CODD79], and "user controllable grouping class" in the Semantic Database Model [HAMM81].

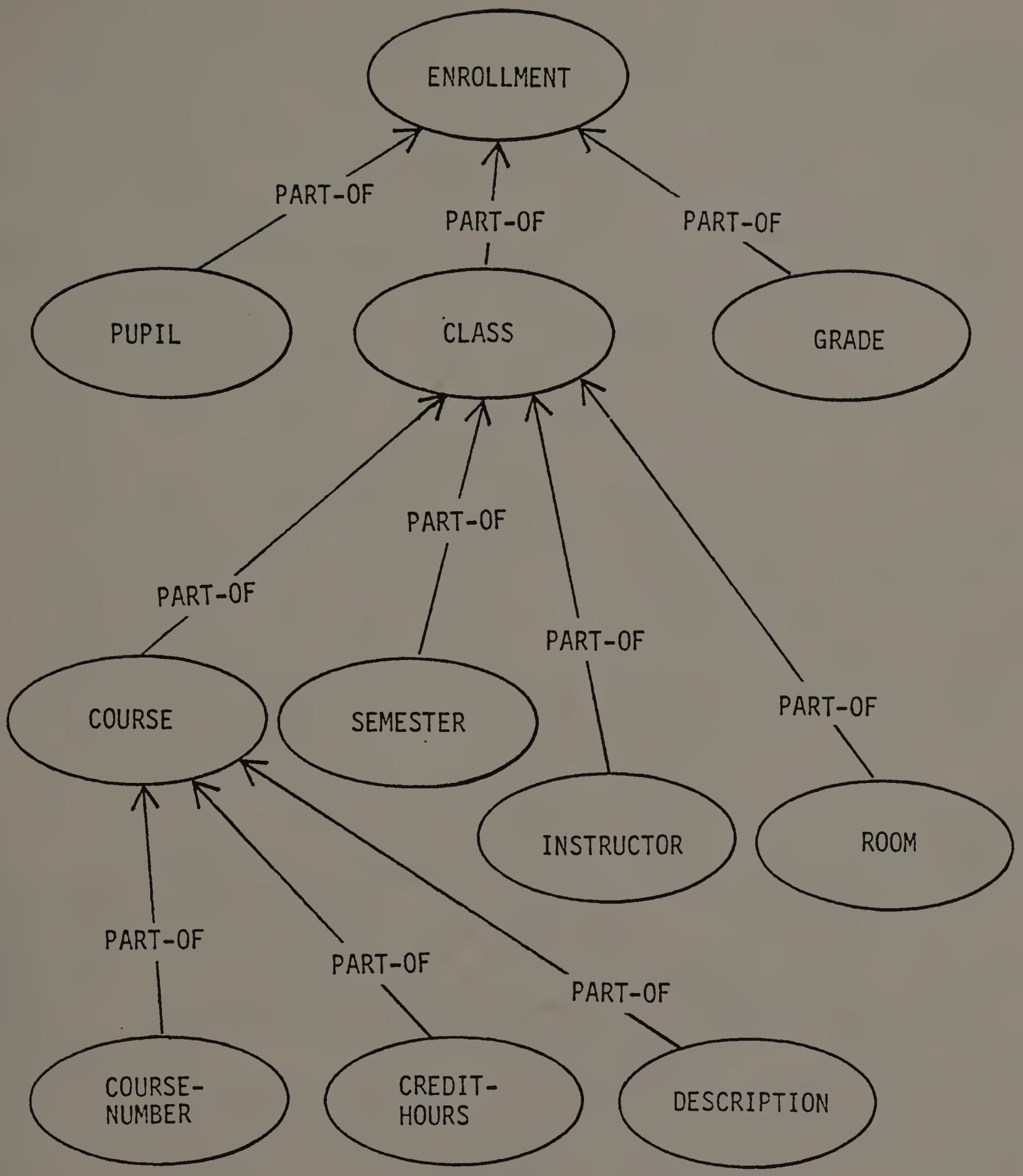


FIGURE 5.2

AN AI PART-OF HIERARCHY

Generalization Abstraction

While aggregation deals with dissimilar objects, generalization abstraction is based on taking advantage of the commonality that exists among objects. Generalization is defined as follows: "A generalization is an abstraction which enables a class of individual objects to be thought of generically as a single named object [italics mine]. Generalization is perhaps the most important mechanism we have for conceptualizing the real world" [SMIT77b, p. 107].

A special case of generalization abstraction is the aforementioned token-type abstraction in which a group of similar elementary objects (tokens) are combined to form a higher level more generic type.

As noted above, the process of forming a type from tokens is called "classification" while the reverse process is "instantiation." More comprehensively, the process of forming a generic type from objects of any abstract level (type or token) is called "generalization." The reverse process is called "specialization" or, alternatively, "generic decomposition."

The attributes of a generic object are those that summarize the attributes of the underlying members of the set. The semantics of this assignment of generic attributes are different than if the attributes were redundantly attached to the members. Generic attributes denote the fact that tokens of the type have these attributes in general. The alternative approach only denotes that those particular tokens hold the attributes.

Generalization abstraction, like aggregation abstraction, is hierarchical in nature [6]. At any level of the hierarchy, the attributes of the ancestor objects are applicable. Therefore, the lower (generic) level objects have a greater number of relevant attributes than do the higher level objects (note that an object holds its own attributes as well as those of the higher level objects).

The fact that lower generic level objects acquire attributes of the higher level more general objects is called the "property inheritance rule" [CODD79]. A general description of this principle is as follows: "Given any subtype e, all of the properties of its parent type(s) are applicable to e. For example, all of the properties of employees in general are applicable to salesman employees in particular" [CODD79, p. 420].

The "attribute inheritance rules" of the Semantic Database Model [HAMM81] deviate from this commonly accepted principle by allowing for parent types to hold attributes which are not inherited by the less general children objects. Such attributes are those that apply to the more general class (type) only, and if inherited would take on an erroneous value and/or meaning. In the above example the attribute "number of employees" is an attribute of the generic object type "employee" and should not be inherited (as is) by the less general type "salesman."

In the AI community, the notion of generalization abstraction goes by the name of the "IS-A" relationship. As is the case with the previously noted AI PART-OF concept, abstraction is depicted by means of linking lower level objects to their higher level objects through binary associations. In this way an IS-A hierarchy, which is synonymous with the notion of a generalization hierarchy, is formed.

Figure 5.3 provides a simple example of an IS-A hierarchy. Note that generalization abstraction is transitive. Thus it is implied that a salesman is a person. In terms of property inheritance, every person has a name, so the attribute NAME is a property of type

PERSON and, accordingly, of types EMPLOYEE, SALESMAN, ACCOUNTANT and DEPENDENT.

Very young persons may not have social security numbers, but all employees do. Consequently, the attribute SS-NO is a property of type EMPLOYEE which is inherited by types SALESMAN and ACCOUNTANT. It is not an attribute of type DEPENDENT, however.

If we consider each non-token object in a generic hierarchy to be the intensional representation of some set of tokens, then we see that the instantiated version of these intensional types can be associated in terms of set based relationships. Specifically, each extension of a lower level object is a (not necessarily proper) subset of the extension of each of its parent objects.

Codd has augmented the notion of hierarchical generalization by specifying two distinct categories of generalization abstraction: unconditional generalization and alternative generalization [CODD79]. The former type corresponds to the standard abstraction concept as heretofore described. The latter type allows for the specification of an exclusive alternation ("X-OR") among a number of supertypes which a single subtype may be generalized into.

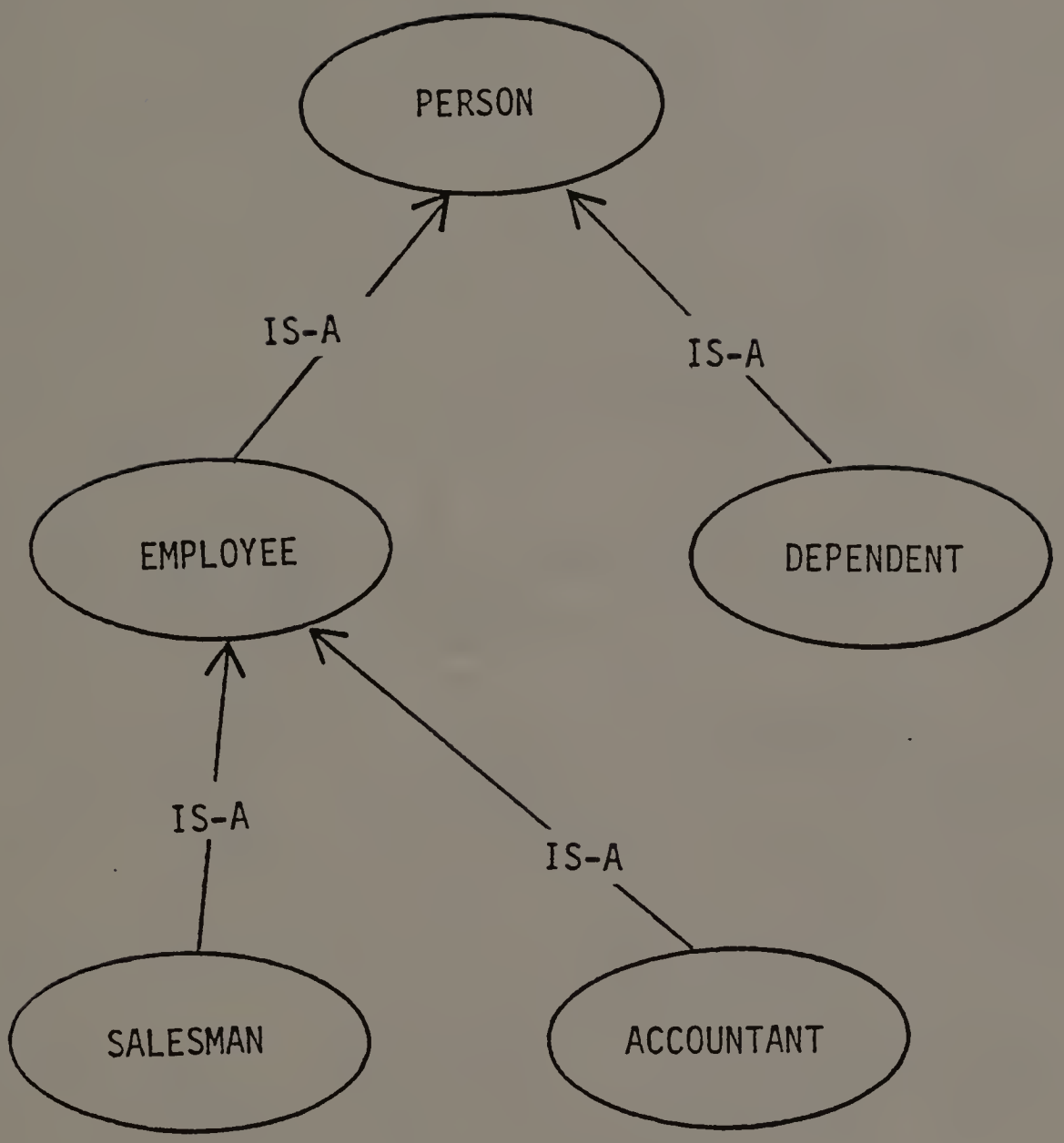


FIGURE 5.3

AN AI IS-A HIERARCHY

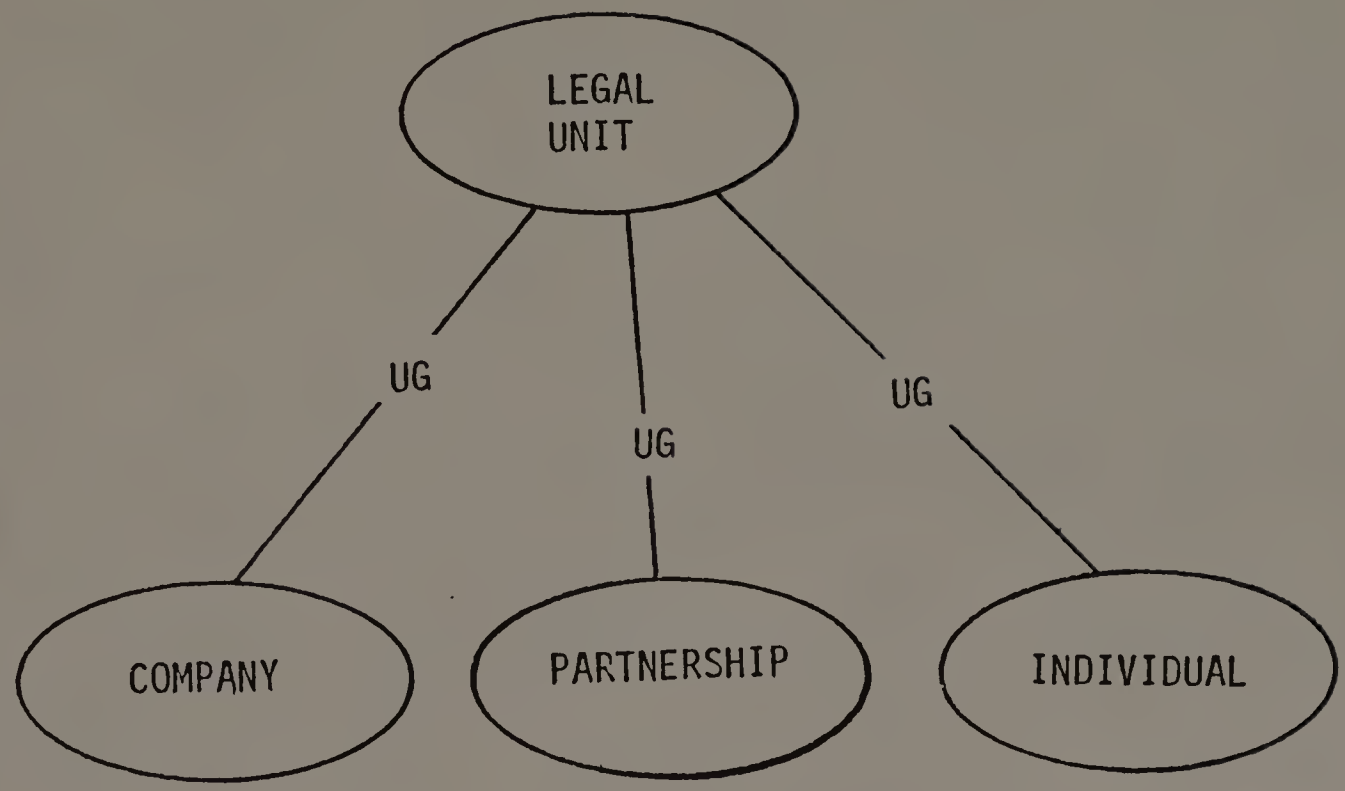


FIGURE 5.4

UNCONDITIONAL GENERALIZATION

Consider, for example, the following five entity types: legal unit, company, partnership, individual, and customer. The type "legal unit" is obviously the most generic concept, so it belongs at the top of the generalization hierarchy. This object is generically decomposed into the three lower (generic) level objects company, partnership, and individual as shown in figure 5.4 (UG denotes unconditional, or ordinary decomposition).

In considering the object type "customer," we find that it is a subtype of the abstract object "legal unit." Also, depending on circumstances, it is a subtype of <u>one</u> of the object types company, partnership, individual. The object therefore belongs below these types in the generic hierarchy with the understanding that a customer token is a member of at most one of its immediate generic supertypes. This semantic is expressed by the alternative generalization concept as shown in figure 5.5 (AG stands for alternative generalization).

Note that a consequence of the alternative decomposition of many abstract objects into a single lower level object is that the tokens of the lower type must be treated individually in terms of property inheritance. All token instances of the type "customer," for example,

should inherit the properties of type "legal unit," but only some tokens should inherit the properties of type "individual" [7].

Discussion

Any abstract object can be decomposed into a set of lower level component objects or into a set of less general objects. Furthermore, these decomposition activities are independent of each other. Consequently, aggregation and generalization can be graphically depicted in orthogonal planes of a three dimensional diagram. A generic class can be decomposed into lower generic level classes by moving in one plane, and each of these objects can be subsequently decomposed into its lower aggregate level objects by moving in the orthogonal plane. This allows for the modeling of quite complex real world phenomena.

In [SMIT77b] there is presented a methodology in which abstract objects can be represented in a relational database. Each generic object is portrayed by an individual relation. A set of five constraints, called the "relational invariants," are used to maintain the

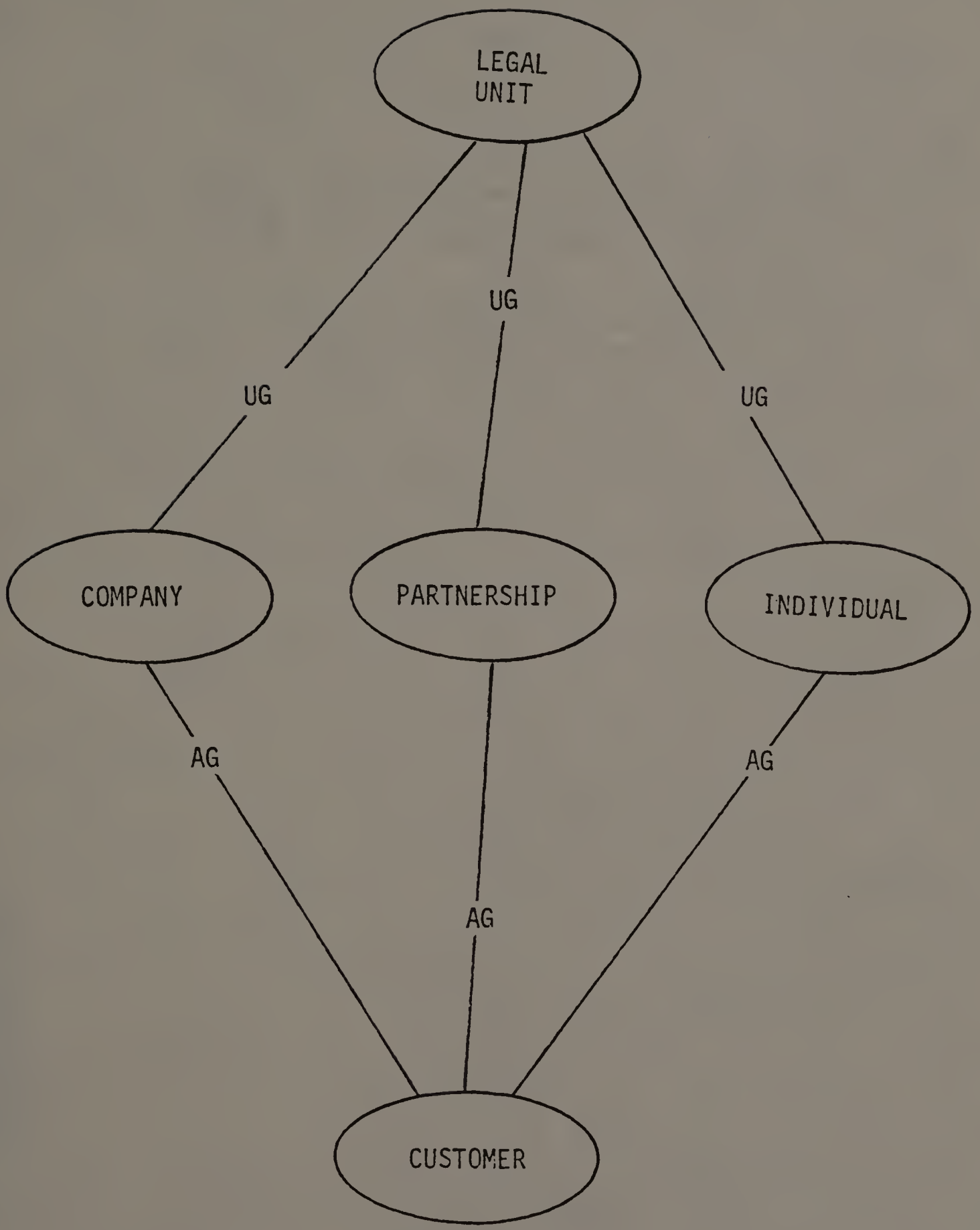


FIGURE 5.5

ALTERNATIVE GENERALIZATION

semantic integrity of the database after update operations are performed.

Satisfaction of the relational invariants constrains each relation to represent an abstract (in terms of aggregation and generalization) object. Because each object is associated with other objects along two orthogonal axes, an update operation which involes an object can trigger the propagation of updates along both the aggregation and the generalization hierarchical structures.

Although they are called by a number of different names, the concepts of aggregation and generalization abstraction are ubiquitous in the fabric of second generaion DMFs. Here, we are not referring to the simplistic notions of token-type generalization or attribute-entity aggregation. Rather, we are speaking of higher level real world abstraction.

The entire Semantic Association Model [SU79] is based on distinguishing various categories of aggregation and generalization abstraction. The Basic Semantic Data Model [SCHM75] is founded on the notion of modeling the aggregate decomposition of objects into component characteristics. Both categories of interclass connection used in the Semantic Database Model [HAMM81] can be viewed

in terms of generalization. The "subtype connection" corresponds to type-type generalization while the "grouping class connection" relates to generic decomposition.

As noted above, the RM/T [CODD79] constructs of "cartesian aggregation" and "unconditional generalization" are synonymous with aggregation abstraction and generalization abstraction, respectively. This DMF distinguishes three types of entity each of which is capable of maintaining its own aggregation and generalization hierarchy.

Perhaps the most enterprising use of abstraction is made by the AI community. In some AI based DMFs "programs," which correspond roughly to traditional database procedures, exist as objects within the semantic net (schema) of the model. As a result, programs can be grouped into classes and related in terms of IS-A and PART-OF hierarchies [TSIC82]. Furthermore, when the standard inheritance rules are applied, programs can inherit attributes from other programs. At the token level we have specific invocations of a program. These are called "processes."

In the next chapter we discuss a number of issues which are related to the implementation of a conceptual data model and data abstraction.

FOOTNOTES

[1] Note that the notions of abstraction and decomposition hold some correspondence to the respective concepts of molecular semantics and atomic semantics as described in [CODD79].

[2] It is obvious that implementation abstraction and physical data independence are closely related concepts.

[3] Note that the database representation of an object is quite often a virtual representation.

[4] It is interesting to note that the relational normal forms [CODD70; CODD72; BEER78] are a formalization of aggregation that is expressed in terms of functional dependency [SMIT77b].

[5] Consider a set S and a family F of subsets sb(1), sb(2) ... sb(n). If each element of S is an element of at least one sb(i) of F, then F is a "covering" of S (see [JAME59]).

[6] Actually, generalization abstraction is network structured. It can, however, be restructured into a tree if token objects are allowed representation in more than one leaf class of the tree (see [SMIT77b] for details).

[7] If this situation is viewed in light of the role model [BACH77], type "customer" is a role type vis. the other entity types.

C H A P T E R V I

SOME IMPLEMENTATION ISSUES

In this chapter some issues which surround the implementation of a conceptual data model are considered. Here, we extend beyond the concept of the model representing the "meaning" of the data -- we discuss how the model affects data storage, usage, and maintenance.

Logical Redundancy and Relativism

The discussion of the modeling of data abstraction, as presented in the previous chapter, gives rise to the need for subsequent discussion of two closely related issues. These are "schema relativism" and "logical redundancy." The former issue has to do with the ability of a schema (and the underlying data modeling formalism) to provide multiple ways of "viewing" the common database. The latter deals with the issue of using derived data to provide such views. Hence, we see logical redundancy as being a means towards an end -- relativism.

Relativist views. Despite the inadequacy of present day programming languages in the provision of primitives to support the representation of real world abstraction [SMIT77a], many database researchers have cited the need for the inclusion of such representations within the constructs of data modeling formalisms [CODD79; HAMM81; SMIT77a; SMIT77b]. It has been further noted that each abstract object, regardless of its level of abstraction, should be uniformly accessible [SMIT77a].

The reason for providing uniform accessibility of all levels of abstraction is twofold. First, such provisions serve to greatly simplify the maintenance of semantic integrity [HAMM81]. Second, and perhaps more important, such provisions aid in the ability of the schema to meet the needs of a greater variety of user groups. The database can be accessed at that abstract level which is best suited for the application at hand.

When considered in this light, we often speak of a data modeling formalism (DMF) as providing a number of different "user views." Each view is based on a specific application. The application "sees" the database through its specific view only and thus, from the perspective of the user group, the data model exists for that application only. The process of providing such application specific

perspectives is called "view modeling" [LUM78]. The ability of a DMF to provide such views is called "relativism" [HAMM81].

Although user views appear to be orthogonal, they are related at both the physical and the logical level. The physical level association stems from the fact that different views share a common set of database objects. The logical level association comes about because the views share a common schema structure.

Consider, for example, that two views access the database at different levels of the generic hierarchy such that the data type used in one view is an (generic) ancestor of the type used in the other view. These views are related at the physical level in that both generic types instantiate into the same token objects. The views are related at the logical level because the generic types involved ultimately exist within a common abstraction hierarchy (see chapter 5). This latter notion is related to the concept of "view integration" [LUM78].

Data derivation. Relativist flexibility demands the presence of logical redundancy in the data model. We say that a schema is logically redundant if some data values are derivable algorithmically from others [HAMM81].

Values which are formed in this way are called "derived data." Data which are not derived are called "base" data. In reference to data access, we often speak in terms of derived versus direct. We will hereunder use the term direct data to refer to data which appears (virtual or actual), upon access, to be base data.

The distinction between derived and base data can be made in terms of individual attributes or in terms of larger database objects. As an example of the former we will refer to the automatic calculation of EMPLOYEE-AGE from DATE-OF-BIRTH and DATE-TODAY. While all three of these items may qualify as direct data, only DATE-OF-BIRTH need be base (we assume that DATE-TODAY is a system supplied function).

As for larger database objects, we call attention to the RM/T "derived relations" which are those that can be derived completely from the underlying "base relations" [CODD79]. From the perspective of the user, both of these relation types hold direct data.

Surprisingly, the concept of a base-derived distinction is not restricted to the datalogical sphere of data modeling (refer back to chapter 3). For example, the nucleus of an infological data model is the set of elementary messages that is physically present (the base

data) [1]. Other messages which are not present can be deduced from the nucleus (see [SUND74], [TSIC82, ch. 11]).

As noted above, abstract conceptual relativism requires that logical redundancy be used; there must be provided direct access to nonbase data. In essence, this means that the DMF should support the modeling of virtual as well as actual data. If virtual data is to be incorporated into the data model, the derivation of derived data must be integrated into the schema. This is the "duality principle" between schema and procedure [HAMM82]. In response to this need the Semantic Database Model includes a set of high level "derivation primitives" through which the specification of virtual abstract data can be made.

Semantic Integrity

Preservation of semantic integrity has to do with the avoidance of errors as to the intrinsic meaning of the database. Although a compromise of semantic integrity may come about as the result of a malicious act, it is most commonly introduced due to a lack of understanding or user error.

In addition to the notion of semantic integrity, three principle aspects of the problem of maintaining database probity can be identified [HAMM75]:

1. Reliability -- avoidance of errors that are due to the failure of executive software (i.e. operating system or DBMS software) and/or the failure or hardware (see [GRAY81], [HONG81], [MARC81], [STEI80], [STON81], [VAND81])

2. Concurrent consistency -- avoidance of errors that are due to the mishandling of shared data by multiple simultaneous users. Maintenance of concurrent consistency revolves around the use of time valued integrity locks, time stamps, and deadlock avoidance algorithms (see [BERN81], [DENN81], [KOHL81], [LEHM81], [RICA81]).

3. Security -- avoidance of database compromise that is due to access by unauthorized users (see [CHIN81], [HARR81], [VAND81])

These database integrity issues, though basally related to the general study of database management systems (DBMSs), are only peripherally related to our task

at hand. They will therefore not be considered further; our concern with database integrity will focus exclusively on semantic integrity.

In terms of semantic integrity constraint specification, the constraints should be generic and intensional vis. token specific. In this way the massive extensional database inherits the limited number of restrictions placed on the intensional types (schema). The reason that semantic integrity is enforced is that it constrains the schema to "more accurately reflect the real world situation" [TSIC82, p. 29].

Inter object dependence. A very basic type of integrity constraint which can be found in a number of DMFs has to do with the interdependence of database objects. This constraint, which is often called the "dependency constraint" [WONG77], refers to the situation where the existence of one object presupposes the existence of another database object.

There are a variety of syntactic dependency constraints which can be identified (in the notation of chapter 2, these are some of the implicit constraints of s(c)). In any tree structure, for instance, the existence of any object (node) depends on the existence of its

entire ancestral chain. Also, it is commonly accepted that the existence of a relationship instance assumes the existence of the token instances which it associates. Similarly, properties are commonly assumed to depend upon the existence of the objects that they describe [2].

Still another example of such relates to the strictly typed property of datalogical DMFs. Since each token must belong to a type, the existence of token objects depends on the existence of the respective type.

We see the above examples as being syntactic because they are structural functions only. In programming terminology they are "in line" issues as opposed to "out of line" issues (see [ELS073]). In other words, there is no evidence of these constraints in terms of the actual execution of a transaction. As such they should not be considered as semantic integrity constraints.

Dependency constraints that are semantic (out of line) in nature are also quite common. In fact, they can be found in some first generation DMFs. The FIXED-MANDATORY and OPTIONAL-AUTOMATIC declarations in the CODASYL DBTG facility have a definite semantic flavor, for example (see [MART75]) [3].

Second generation examples of semantic dependency are abundant. In the Entity-Relationship Model [CHEN76] for instance, the existence of "weak entities" and "weak relationships" are dependent on the existence of other objects. The "characterizing entities" of the Semantic Association Model [SU79] depend on the existence of the "defined entities" with which they are associated. Also, when objects participate in a "depending relationship" within the Basic Semantic Data Model [SCHM75], the existence of the "depending part" relies upon the existence of the "ruling part."

It is noted in [WONG77] that use of the IS-A relationship in artificial intelligence (AI) applications (and consequently use of generalization abstraction in database management (DM)) helps to guarantee database consistency; it defines the "IS-A constraint." Addition and deletion of token objects of one type are automatically carried to all generic supertypes. Since this automatic modification occurs for semantic vis. structural reasons, we see the IS-A constraint as helping to maintain semantic integrity. This notion relates closely to the relational invariant based update triggers [SMIT77b] as described previously in chapter 5.

An integrity subsystem. There can be defined two major approaches to semantic integrity specification: static and dynamic. These are called the "state snapshot approach" and the "state transition approach," respectively [HAMM75].

In the state snapshot approach to semantic integrity, rules are used to specify the valid states (database occurrences) that the database may take on. If the dynamic nature of a database is thought of as being represented in a continuous motion picture, then the integrity rules are specified such that each frame of the movie provides a semantically correct picture. The concept of a database state snapshot is the datalogical equivalent of a "time slice" [BILL78; SUND73] as used in the infological realm of data modeling.

The state transition approach to semantic integrity deals with constraining operations that are performed on the database. Integrity rules define the legality, depending on present database state, of operations that convert the database to new states. If an operation is legal (called a valid operation under the current state), then preservation of the integrity of the database is guaranteed under the operation.

Hammer and McLeod have proposed a "semantic integrity subsystem" in conjunction with their efforts on the IBM supported MIT Project MAC [HAMM75]. This subsystem, which represents an integration of the snapshot approach and the transition approach, is briefly described below in terms of a basic relational implementaion. Because of its modularity, the concepts are extendible to any modeling formalism.

Many of the constructs upon which the methodology is based correspond to the three major categories of constraints as described in [CHEN76]. These three categories are as follows:

1. Constraints on "allowable values"

2. Constraints on "permitted values"

3. Constraints on "existing values"

The parallelism of these constructs to those in [HAMM75] will be noted when appropriate.

A basic premise of the integrity subsystem is that there must be defined two types of constraints: domain constraints, and relation constraints. Specification of a domain constraint involves precise description of the set

of atomic objects that constitute the domain; it is a domain definition process.

Each domain referenced is associated with a constraint specification. These domain constraints correspond to the constraints on allowable values of [CHEN76]. At a minimum, the objects of a domain must be constrained to belong to one of two mutually exclusive categories. These categories relate to the natural data type primitives: string and number.

Relation constraints have to do with restricting the ways that relations or subparts of relations may be related. Relation subparts can be tuples or columns, and their associated constraints can be inter- or intra-relational in character. Such constraints are predications on the state and/or transactions of the database [4].

In the event that a relation subpart is a column, the respective constraint corresponds to Chen's permitted values. Tuple oriented constraints relate to the restrictions on existing values of Chen.

The proposed constraint management facility comprises a number of components:

1. A high level constraint specification facility for each of domain constraints and relation constraints

2. Processors for the domain and relation constraint specification languages

3. A constraint checker

4. A violation-action processor

5. A constraint compatability checker

The use of high level constraint languages allows for the facility to take advantage of the inherent logical structure of the underlying data modeling formalism. It is suggested that these languages be declarative vis. procedural oriented. Each of the two languages allows for the specification of the "nature," "enforcement," and "violation-action" of each constraint.

Nature and enforcement define what the constraint is and when it is to hold, respectively. Violation-action specifies the system's response to occur in the event that violation of a constraint is detected.

The language processors convert the high level language to the appropriate internal form that is used by the integrity management facility. The function of the constraint checker is to determine when checks are appropriate and to perform these checks.

In the event that a constraint violation is found, the violation-action processor initiates the appropriate action. This action is described in the internal form representation of the violation-action specification of the constraint as noted above. Said action may involve the reporting of the occurrence of the semantic error and/or the inititation of some automatic corrective action.

It should be noted that automatic correction of semantic errors may be hazardous because of the dependence of relation constraints upon context. Replacement of some erroneous value with a system specified value can trigger the propagation of a chain of semantic violations throughout the database. An obvious "safe" corrective action is the replacement of the item in question with the value NULL [5].

The constraint compatability checker, which is undoubtedly the most difficult component to implement, has the responsibility of detecting circularities and logical

conflicts from among the set of constraints that is specified. The complexity of this component is due largely to the wide potential scope of corrective action side effects.

Because relation constraints are relatively complex objects, their description deserves further comment. First, proper specification of a relation constraint demands precise description of the "constrained" (focus of the constraint) as well as of the "constraining" (properties of the constraint). The constrained is defined in terms of "specification operations" (see [TSIC82]) which describe, either explicitly (local) or implicitly (global), the scope of the constraint. Since relation constraints are context dependent, their delineation relies heavily on the notion of "data relatability" in terms of closed-form formulation (see [TSIC82] especially chapter 4).

The constraints can be either aggregate, in which the constrained object is atomic, or nonaggregate. Aggregate objects are based on some single property of a relational attribute such as "sum," "average," "count." Nonaggregate column constraints can be functional (mapping) constraints, or structural constraints.

A Brief Recapitulation

The text to this point has provided an overview of the general concepts and principles of conceptual data modeling. Both theoretical and pragmatic issues have been discussed in detail. The interested reader is directed to appendix A in which the major approaches and schools of data modeling are described.

FOOTNOTES

[1] An infological "elementary message" is a reference to a "constellation." A constellation is a triple (a,b,c) which represents a basic fact. In this triple, a is a tuple of objects, b is a relationship among or a property of the objects, and c is a time specification. Note that an elementary message is a reference to a constellation rather than the constellation itself. This distinction is similar to that between call by name and call by value as made in programming.

[2] An interesting exception is found in the Semantic Binary Model [ABRI74]. Here, an object can hold an association with an "unknown" (null) object so that the existence of the association does not require the existence of all constituent objects.

[3] The semantic of MANDATORY membership is that the lifetime existence of the object depends on its membership in at least one set of the specified type. The semantic of AUTOMATIC membership is that object creation depends upon a priori set membership.

[4] In terms of logical complexity, the invocation of relation constraints is much more involved. They are defined with respect to relationships between objects and, as such, are context dependent. The simpler domain constraints, on the other hand, are context free.

[5] In regards to database semantics the meaning of the value NULL can have widely different interpretations depending on whether the implementation adopts the "open world" or the "closed world" view (see [VASS79], [REIT78]).

PART II

THE NATURE OF GEOPOLITICAL STATISTICAL DATA -- STRUCTURE AND SEMANTICS

"In an imperfectly organized system, even if every part performs as well as possible relative to its own objectives, the total system will often not perform as well as possible relative to its objectives."

Russell L. Ackoff
"Towards a System of Systems Concepts"

C H A P T E R V I I

GEOPOLITICAL STATISTICAL DATA: PHYSICAL STRUCTURE

Introduction

In the view of the database management community proper utiliization of a data set requires the presence of two distinct structures: the physical structure and the logical structure. Physical structural issues relate to the formation of data records from data items, the definition of record types, and the structuring of record types to form files. Logical structural issues relate to the arrangement of the database primitives (entities, relationships, properties, values) in the formation of a logical model of the relevant environment.

When a logical data model is used in accessing the contents of a specific data set, as is the case here, it is called a schema of the data set. In the remainder of this work we will use the terms data model and schema interchangeably since it is understood that each data model is intended for use with a data set.

A brief review of the nature of the four database primitives is due here. Entities represent the real world objects with which the data user is ultimately concerned. Relationships are assertions about associations among entities. Properties are assertions which characterize entities.

The model provided by the schema does not deal with the individual entities. Rather, it depicts assertions about entity types (database intension). When the properties of entity types, as defined by the schema framework, are "filled in" with data values specific entities are represented. Thus values, though fundamentally related to the database contents, are indirectly related to the structure of the data model.

A conceptual schema is one which captures the semantics of the underlying data set. Again, as outlined in chapter 3, data semantics are the "meaning" of data [KERS76]. The reason behind our concern with data semantics is that they "ensure that stored data more accurately represents the enterprise [application environment]" [BEER78]. Here, the term "more accurately" can be interpreted as meaning "closer to our cognitions and perceptions." A conceptual schema, then, illustrates both the contents and the meaning of the data set.

For obvious reasons the conceptual schema should provide an accurate "picture" of reality. An accurate schema supports a proper understanding of the database contents on the part of the user. The entity types depicted in the schema should closely resemble the "kinds" of objects that exist in the relevant environment. The relationships of the schema should match the important associations that exist among the actual entities. The properties of schema entity types should relate to those characteristics of real world entities that are important within the logical framework of the application.

The logical structure of real world objects is hardly simple. An accurate conceptional schema, then, demands complexity in its structure. In terms of data processing, however, complex structures result in processing inefficiencies. When designing physical data structures good engineering is simple engineering. As a consequence, the design criteria of the logical structure conflict with those of the physical structure. Objects structured with processing efficiency in mind provide the user with a poor logical model of the world. Objects structured to match the nature of reality are difficult to process.

A solution to this dilemma lies in the principle of physical data independence. Under the precept of data independence the logical model of the data set is designed independent of physical data structural considerations. This separation of logical and physical issues allows for the occurrence of the following situation: the physical structure exists to satisfy the specific needs of automated data processing while, simultaneously, the logical structure (conceptual schema) exists to satisfy the specific needs of the user community.

One important consequence of achieving physical data independence is that the record types of the physical structure need not match the logical entity types of the conceptual schema. A second result is that the connective links (pointers, key chains, etc.) of the physical structure need not be synonymous with the relationships depicted in the conceptual schema.

From this point on we will be concerned with a particular type of data -- geopolitical statistical data. The uses and users of such data were described in chapter 1. The majority of applications of data of this type revolve around the use of census data sets. Therefore, the issues discussed hereunder will deal, almost entirely, with census data. The reader should keep

in mind, however, that the tools and techniques developed in this work are applicable to any statistical data which is geographic based.

At present, conceptual schemas for geographic based statistical data sets do not exist; the data hold a physical structure only. Also, there is no data modeling formalism which supports the creation of appropriate logical models of such data. As a consequence of this, the physical data structure is often used to serve as the logical model of the underlying data.

The records and links of the physical structure, which are designed for efficient automated data processing, are the objects with which the user must deal. A record type, viewed in terms of real world geographic entity types, may be void of any logical significance. Connective links between records do not necessarily relate to a meaningful logical relationship between the connected objects.

In short, the physical data structure of census data files provides a poor logical model of the underlying data. From the point of view of the data user the structure neither furnishes an understanding of the meaning of the data nor involves logically meaningful objects.

Later in the discussion we will develop a data modeling formalism which deals with the relevant logical objects (database primitives) of geopolitical statistical data sets. First, however, we will investigate the physical structure of census files and the problems that arise from its use as a logical model.

The Physical Structure of Census Files

Tree structures are used in a variety of database management and data processing applications. Census data files are ultimately based on tree structures, so we feel that the general structural form deserves comment here.

It is assumed that the reader holds an a priori understanding of the tree as a data structure. The terminology that is used by the database practitioner tends to be somewhat loosely defined, however. For this reason, we will provide a brief review of tree structure terminology as it applies to our work. A more rigorous treatment of this data structure is found in [HORO76] and in [PAGE78].

Trees and hierarchical files. A tree is a hierarchy of nodes which is structured so that each node is linked to exactly one node in the next higher level of the structure. The very top node is called the "root" of the tree. The very bottom nodes (more generally the nodes which are connected from above only) are called the "leaves" of the tree. Note that, metaphorically speaking, the structure would be better called an inverted tree as the branches "grow" down rather than up.

At each level of the structure the element in the next higher level to which a node is connected is called the "parent" of the node. Similarly, at each level of the tree, the element(s) in the next lower level to which a node is connected are called the "children" of the node [1]. Obviously, the root of the tree has no parent and the leaves have no children.

Nodes which share a common parent are called "sibling" nodes. For any node the elements which form the path to the root, which incidently is a unique path, constitute the "ancestry" of the node. Thus, the ancestry of a node consists of the parent element, the parent of the parent (grandparent), the parent of the grandparent (great grandparent), and so on up to and including the root. Since siblings have a common parent, and the path to the root node is unique, they share a common ancestry.

Hierarchical files. The records of a file can be structured to form a tree. In such a case the file is called a "hierarchical file." Census data files are hierarchical files, so we can think of census data records as having parent, children, and sibling records.

In a hierarchical file all instances of a given record type exist at a single level of the hierarchical structure. In the case where all elements of a hierarchical level are restricted to be instances of a single record type, the record structure is a "non-branching hierarchy" (the record types do not branch but, obviously, record instances do).

A less restrictive hierarchical file structure is one in which a level of the hierarchy is allowed to include instances of more than one record type. Such a file structure is called a "branching hierarchy." Census data files, in general, form branching hierarchies.

Figure 7.1 provides an example of a branching and a non-branching hierarchical file structure. Note that each rectangle in figure 7.1 represents a record type as opposed to a specific record instance. It is the instances of these types which are tree structured.

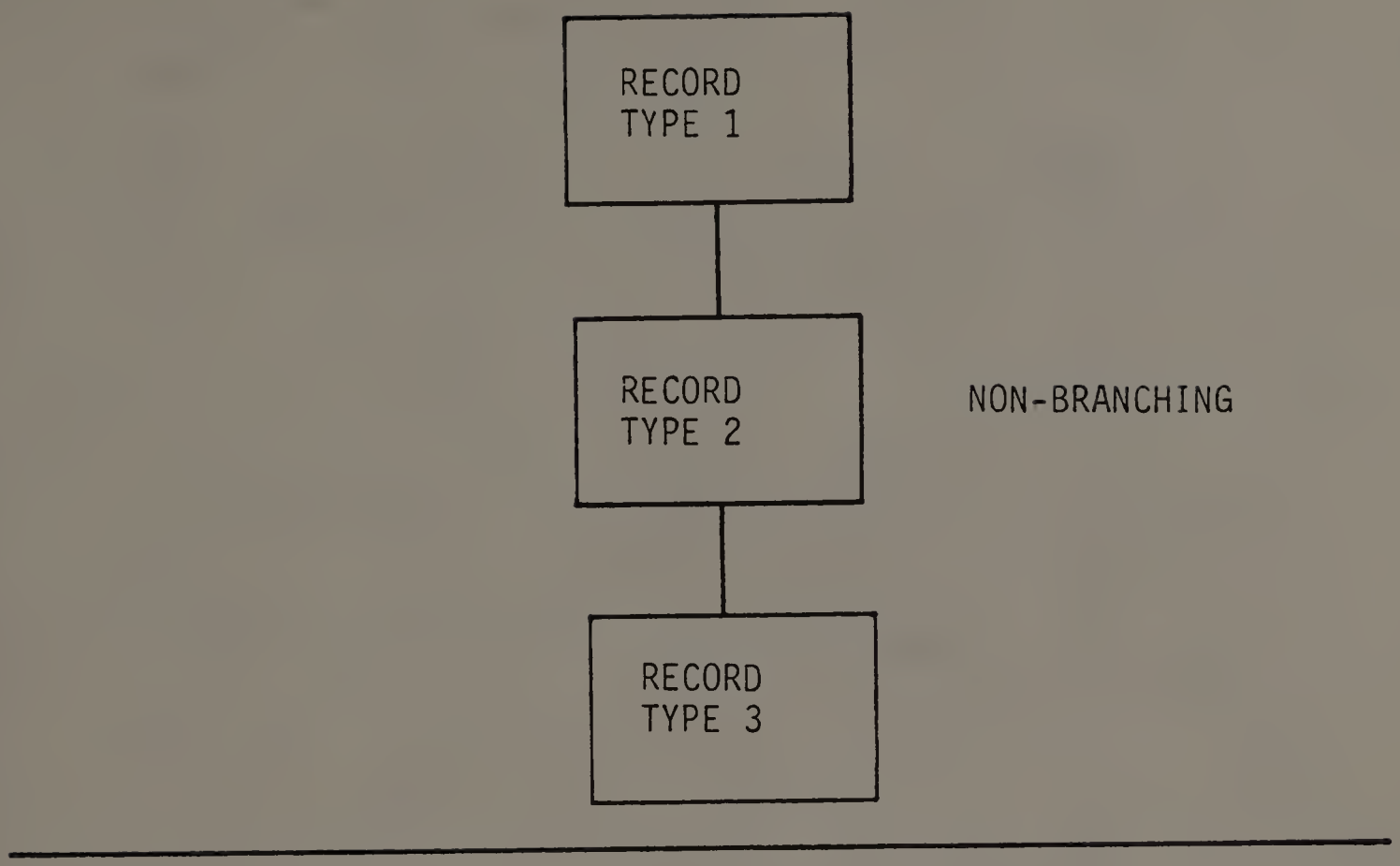

RECORD
TYPE 1
RECORD
TYPE 2
RECORD
TYPE 3
NON-BRANCHING
RECORD
TYPE 1
RECORD
TYPE 2
RECORD
TYPE 3
RECORD
TYPE 4
RECORD
TYPE 5
RECORD
TYPE 6
BRANCHING

FIGURE 7.1

It is important to realize that, in both the branching and the non-branching case, the instances of a given record type exist at a single hierarchical level.

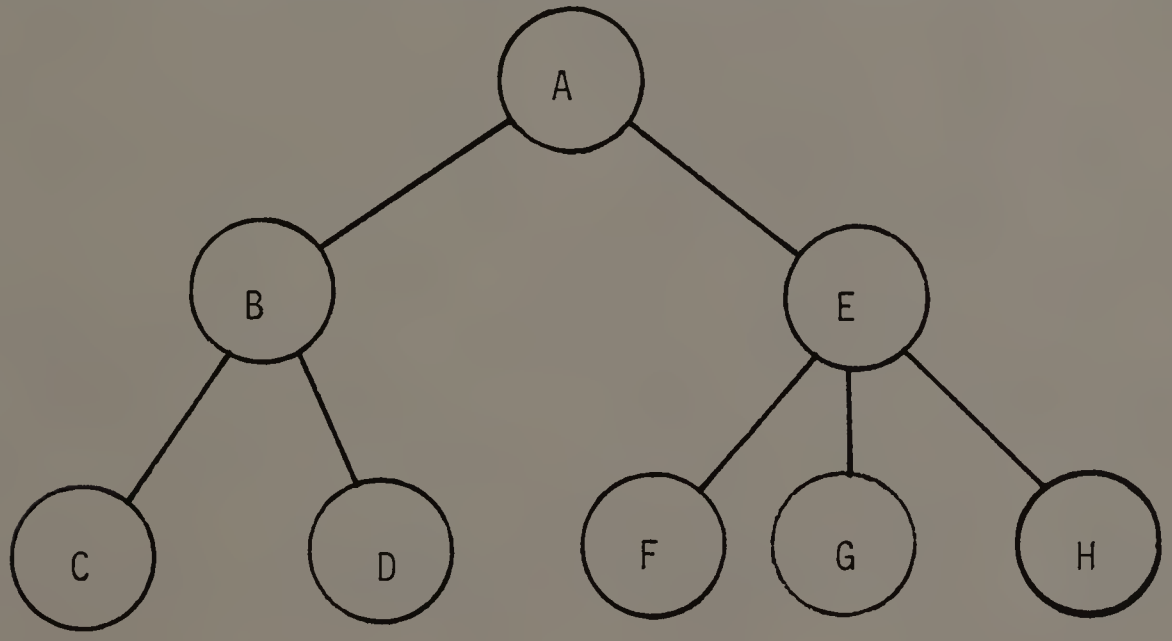
There are a number of ways that the individual records of a hierarchical file can be structured to form a tree. The record structure can be specified through the use of pointers, record position, or record key (see [MART75]). In census data files the record keys are used to define the tree structure.

Briefly, under this method an element of the tree structure is identified by the concatenation of the value of its key with that of the key of its ancestry. The primary key of a record uniquely identifies it among the instances of the record type. The primary key of a record, coupled with the concatenation of the keys of its hierarchical ancestors (called the key of its ancestry), uniquely identifies the record within the total tree structure. Each record of a census data file therefore carries its own key value as well as the key values of its ancestor records.

Figure 7.2 illustrates this method. Note that, for the sake of simplicity, the example represents a non-branching hierarchy. The technique is directly applicable to hierarchies of the branching variety.

RECORD	RECORD TYPE	TYPE 1 KEY	TYPE 2 KEY	TYPE 3 KEY	DATA
A	1	01	---	--	xxxx
B	2	01	001	----	xxxx
C	3	01	001	0001	xxxx
D	3	01	001	0002	xxxx
E	2	01	002	----	xxxx
F	3	01	002	0003	xxxx
G	3	01	002	0004	xxxx
H	3	01	002	0005	xxxx

RECORD INSTANCES



RECORD INSTANCE STRUCTURE

FIGURE 7.2

Up to this point we have described the nature of the hierarchical structure upon which census data files are based. The reader is reminded that hierarchical files are defined in terms of records and record types. There has been no mention of the relationship between a record and a logical geographic entity nor has there been drawn an association between record types and geographic entity types.

The physical structures of census files are defined hierarchically because hierarchical files are conducive to efficient data processing. Since there does not exist a logical file structure defined with the user in mind, the hierarchy serves as both the physical structure and the logical file structure. Thus the records of the physical structure are the objects with which the user must work. Also the hierarchical associations (connective links) among record types serve as a model of the logical geographic relationships.

We will soon see that adherence to the highly restrictive (physical) tree structure can cause the records of the data file to be void of any logical significance. Furthermore, we will see that a hierarchy provides a poor model of the logical relationships that truly exist among geographic areas.

Conceptual Problems

In geographic hierarchical files each level of the hierarchy relates to a different partitioning of the common area (e.g. a state) from which the data are collected. The partitioning at each level is based ultimately upon well defined geographic objects (appendix B defines many of these objects). For example, at one level the area may be partitioned according to counties, while at a lower level the area may be partitioned according to census tract. As an example we will use the 1980 version of the Bureau of the Census Summary Tape File-1B. For brevity, the abbreviation STF1B is used hereafter to specify this census file.

The physical structure of STF1B is such that the records form a hierarchy (tree) of geographic based summary data. The file contains seven distinct record types each of which carries the data for a specific type of geographic area. The records are hierarchically "nested" such that, in all cases, the lower level (children) areas are subsumed by the higher level (parent) areas.

STF1B records are nested in the following way (successive indentation indicates successive nesting):

STATE

SMSA

COUNTY

MCD

PLACE

TRACT or BLOCK NUMBERING AREA

BLOCK or ENUMERATION DISTRICT

Thus, county records are nested within SMSA records which, in turn, are nested within the state (root of the tree) record. Note that STF1B forms a non-branching hierarchy unlike many census data files.

The non-containment problem. When geographic based statistical data is provided as a tree structure such that the children records are subsumed by their parent record, a number of anomalous situations can arise. First, consider the case where the geographic boundaries of an

instance of a child area transcend the boundaries of the parent area. In other words, the lower level area is divided by the higher level area.

Given the physical structure of STF1B a situation of this type can (and does) arise at a number of levels of the file hierarchy. For example, a standard metropolitan statistical area (SMSA) is defined as a set of physically contiguous counties which exhibit a high degree of economic and social integration and which display certain population characteristics (see [KAPL80]). With this definition it is obvious that there exists a natural hierarchical relationship between SMSAs and counties.

A problem arises, however, because SMSAs are nested within state in the STF1B configuration. The counties which form an SMSA are not constrained to lie in the same state and thus an SMSA will often be divided by the boundaries of a state. Similar situations arise at a number of levels of STF1B. The issue will hereafter be referred to as the "non-containment problem" of a geographic hierarchy.

When record types are nested to form a tree structure as in STF1B one can think of the child records as being "owned" by their (single) parent record. In the above noted situation of a "multi-state" SMSA, the

geographic object is the logical child of two state parents. In reality, then, a network (or plex) structure is required to represent the true logical state-SMSA association (see figure 7.3a).

The Bureau of the Census has chosen to address the non-containment problem by introducing the concept of "part of area" records. When a ggeographic object is divided by a higher (tree) level entity, a representative record for each portion of the divided object is included in the file. These "part of area" records are nested within the appropriate higher level area records. Note that the parts of a divided area are not siblings in the physical tree structure; they are of different parents. Intuitively, then, we can say that the parent area "owns" that portion of the child area that lies within its geographic boundaries.

The Bureau of the Census solution to the non-containment problem allows for the hierarchical physical representation of the geographic data. The technique may have detrimental effects upon the information retrieval process, however. To the data user the appropriateness of this methodology as a solution to the non-containment problem is debatable. Specifically, the suitability of the method is dependent on the primary

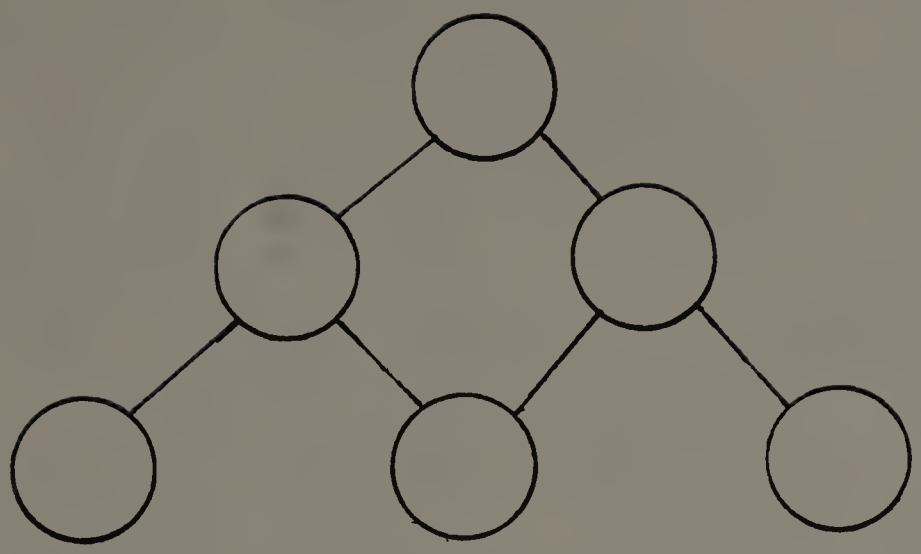


FIGURE 7.3a

THE STATE-SMSA
NON-CONTAINMENT PROBLEM

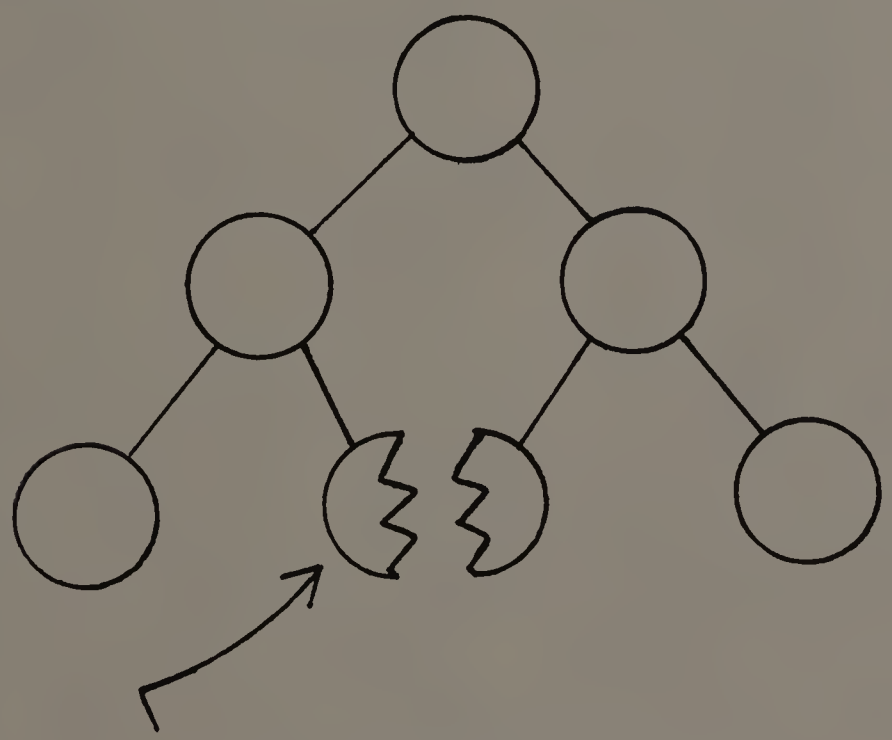


FIGURE 7.3b

THE CENSUS SOLUTION

geographic object in which the user of the information is interested.

In the state-SMSA non-containment situation the user who is primarily interested in state as a unit of analysis would find the situation satisfactory. State, as a geographic object, is unaffected by the splitting of SMSAs. On the other hand, however, if the analysis is SMSA oriented, maintenance of the hierarchical structure yields an added (and unintuitive) complexity to the information gathering process. The user is forced to deal with an unnatural "part of" geographic object -- an entity which arises as a side effect of attempting to maintain a chosen physical file structure (see figure 7.3b).

The non-coverage problem. A second anomaly that stems from the hierarchical presentation of geographic based data will be called "non-coverage problem" of a geographic hierarchy. The non-coverage problem arises when the child area(s) of a hierarchical relationship fail to provide complete geographic coverage (called grid coverage) of the parent area which owns them.

As an example, let us again consider the top three levels of the STF1B geographic hierarchy: state, SMSA, and county. For the sake of simplicity we will ignore the

non-containment problem and assume that SMSAs do not transcend state boundaries.

Now, in recalling the previously cited definition of SMSA, it should be obvious to the reader that the set of SMSAs within a state would not be expected to completely cover the state. If we wish to maintain the state-SMSA hierarchical relationship, then we must choose one of two possible courses of action:

1. Include a (child) record for each SMSA that lies within the (parent) state and leave the non-SMSA areas unrepresented at this level of the hierarchy

2. Include a (child) record for each SMSA that lies within with (parent) state and, in addition, include a complementary record representing all areas of the state which are not covered by SMSAs

Upon inspection, the reader may find the former alternative, including true SMSA records only, the more intuitively pleasing choice. With regards to the data needs of the user the SMSA level of the geographic hierarchy should not be confounded by non-SMSA information. In considering the lower hierarchical

levels, however, we see that this choice would disrupt the nature of the tree structure.

Counties, in all cases, completely cover their state; SMSAs do not. As a result, there are some counties which are not part of an SMSA. Maintenance of the hierarchical nesting of counties within SMSA within state, as seen in STF1B, would define "orphan" counties under alternative 1. All non-SMSA counties would have no parent in the successive level of the physical structure. This is illustrated in figure 7.4a.

The second option, providing a SMSA-complement area record is the method which is used by the Bureau of the Census. All non-SMSA county records are owned by the complementary record. In file documentation provided by the Bureau of the Census this record is referred to as a "pseudo SMSA" or "remainder of state" record. We feel that the intuitive appeal of this methodology is less than optimal from the logical perspective of the user. True, the physical hierarchical structure of the file is maintained, but access to the data requires that artificial geographic objects be maintained, processed, and understood by the user (see figure 7.4b).

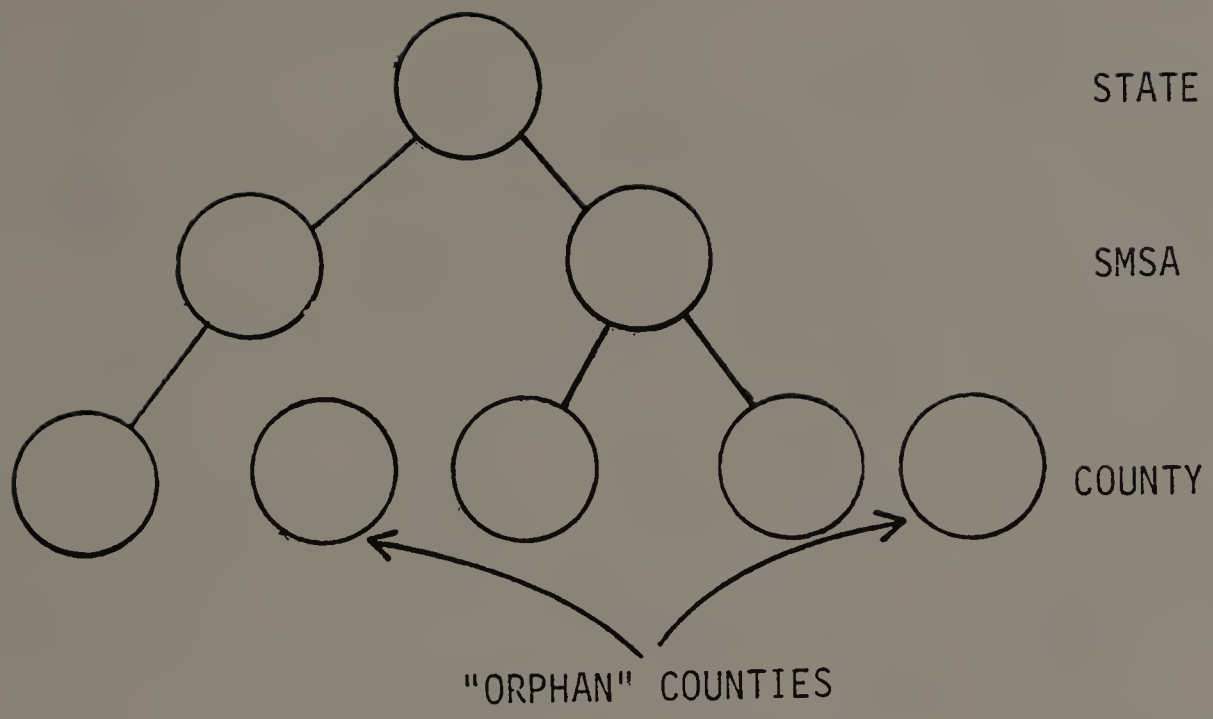


FIGURE 7.4a

THE STATE-SMSA NON-COVERAGE PROBLEM

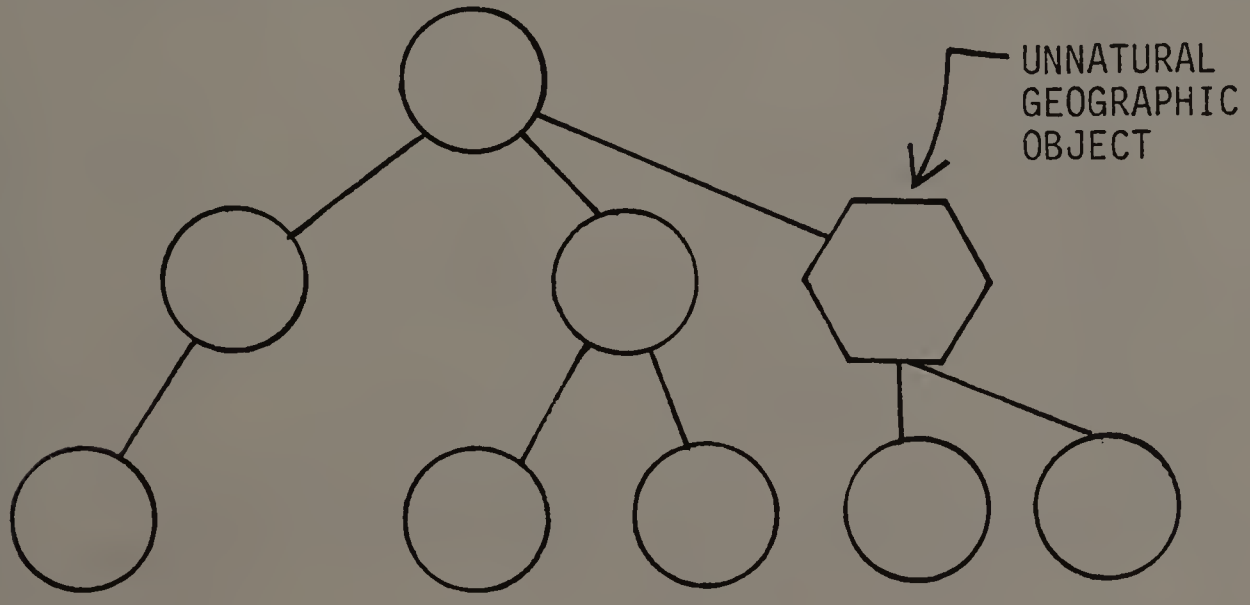


FIGURE 7.4b

THE CENSUS SOLUTION

As is the case with the non-containment problem, the non-coverage of parent by children areas exists at numerous levels of the STF1B hierarchy.

Multiple complexities. One attribute of a physical hierarchical file structure of the type discussed here is that an object within the structure is affected by its ancestry. Each record carries the identifiers (keys) of all superordinate owner records. In fact, each record is identified by the concatenation of the keys of its ancestor records.

When working with geographic hierarchies, then, identification and understanding of a geographic object instance requires that all ancestor instances in the hierarchy be considered. A geographic artificiality at level M of the hierarchy therefore results in the propagation of anomalies at each N (>M) level of the structure (note that the level numbers increase as we move down the tree).

Recall that both the non-containment problem and the non-coverage problem were said to exist at a number of levels of the STF1B hierarchy. As a result, geographic object instances at the lower structural levels can be plagued by multiple complexities in their identification.

An STF1B block instance, for example, is identified by the concatenation of seven geographic area key values (one taken from each level of the structure). This provides the potential for more than one "non-entity" (pseudo or part of area) to arise within the definition of a block instance. Obviously, situations of this type act as confounding factors in the information retrieval process and can only serve as a hindrance to a proper logical understanding of the geographic information.

As testimony to the confusing nature of the contents of geographic hierarchical files consider the definition of a level 6 STF1B record. The geographic object represented by a record at this hierarchical level is defined as follows: "Tract (or Block Numbering Area) or portion of Tract (or Block Numbering Area) within place, place segment and remainder of county or MCD" [CENS80, p. 28].

In the previous paragraphs we outlined two anomalous situations which often arise when geographic data are structured as a tree. If a hierarchical structure is to be maintained, the non-containment problem requires that well-defined meaningful geographic areas be divided and represented via "part of area" records. These geographic areas are void of any logical significance <u>when</u> <u>viewed</u>

outside the physical hierarchy in which they exist. A given set of geographic objects, when structured in two different hierarchical orderings, will result in two completely different dissections of geographic area instances.

The non-coverage problem is also guilty for defining the need for meaningless geographic areas. The "pseudo area" complementary records have no semantic foundation and exist merely as a structural implementation tool.

The point to be made from the above discussion is this: geographic entities do not hold a natural hierarchical ordering. Therefore, implementation of a geographic hierarchy requires that the objects be "forced" into an artificial situation.

The reason behind the non-hierarchical nature of geography is simple -- different geographic partitionings are defined for different purposes. Geographic areas may be governmental units, statistical units, or enumeration units. The significance of any one of these geographic area types depends upon the specific application in which the data are used. The criteria used to define these widely different area types are outlined in [KAPL80].

Misrepresented relationships. The non-containment and non-coverage problems are important because they disrupt the nature of the geographic objects that are represented by the underlying data. These issues force the user to deal with geographic objects which may have no logical significance; records do not relate to meaningful entities.

Beyond this, the hierarchical physical structure fails to provide an accurate conceptual model of the data by misrepresenting geographic relationships. The nesting of county within SMSA within state, as in the STF1B configuration, holds the logical implication that SMSA geographic objects are more closely related to state objects than are counties.

Furthermore, the chosen configuration implies that the "importance" of an SMSA area is somehow subordinate to that of a state area. We suggest that the logical significance of these area types cannot in general, be specified; it is dependent upon the needs of the user.

In short, the very restrictive structure of a tree is not well suited to represent the logical associations that exist among widely heterogeneous geographic areas. The definition of political areas is usually independent of statistical considerations. Similarly, large political

areas are seldom considered in the definition of statistical zones. Why then should there exist a logical "parent-child" structure among such areas?

In common business database applications schemas are often tree structured because the logical relationship among the entities of the application are truly hierarchical in nature. For example, the fact that an employee entity "owns" its dependent entities denotes, to the data user, a logically rational relationship among the objects. The same cannot be said about the objects of a geographic database.

Attribute areas: representing entities as properties. Still another inadequacy in the ability of the physical structure of census data files to serve as a proper conceptual schema lies in the makeup of the individual record types. Recall that each record type of the file relates to some partitioning of geography. Each record instance, then, denotes some (not necessarily meaningful) geographic area whose properties are specified within the attributes of the record.

The geographic area types that correspond to the record types are used to form the "nested" geographic hierarchical file structure, so we will refer to such area

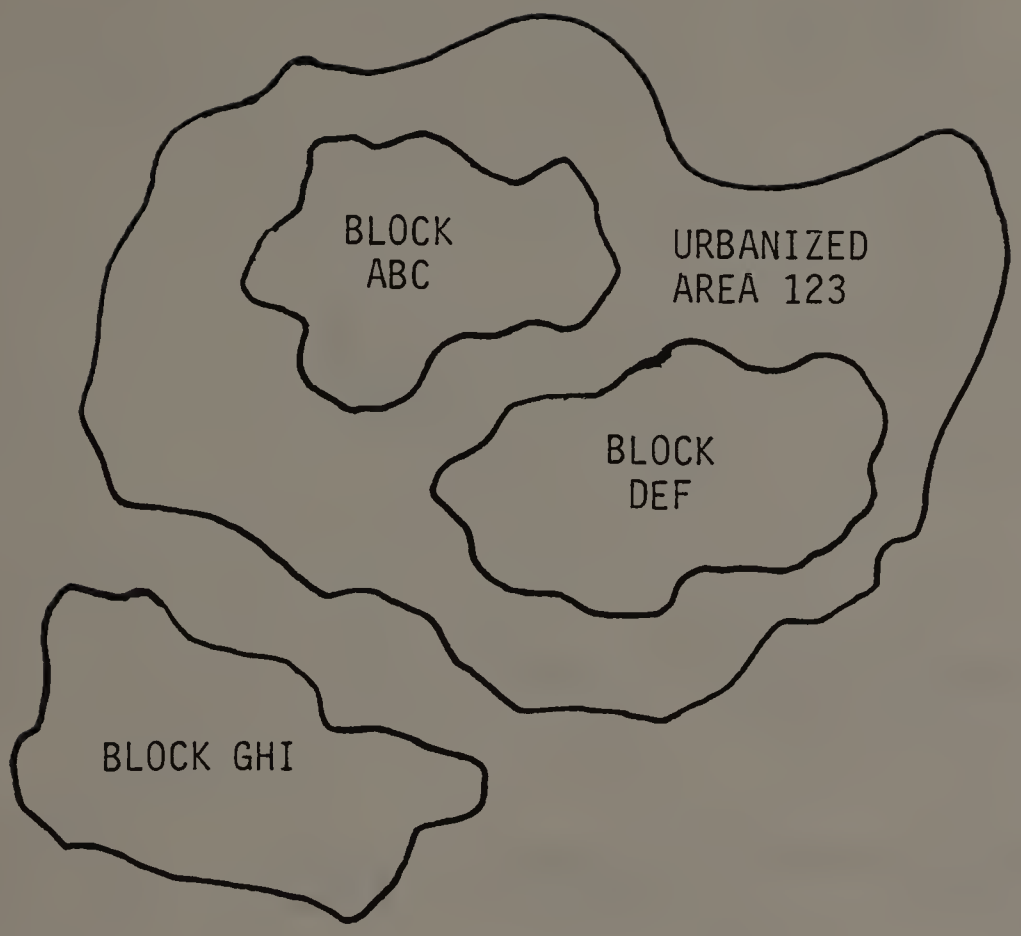
types as "nesting areas." Example STF1B nesting areas are state, county, and SMSA.

A problem is that not all relevant geographic area types about which information is provided in the file hold positions as levels of the hierarchy. In other words, census files contain information about geographic areas which are not nesting areas. We will refer to such geographic object types as "attribute areas."

An attribute area is described when the geographic (nesting) area instance represented by a data record is "flagged" as being part of another area type. For example, a specific STF1B record may represent an instance of a census block where the area type BLOCK assumes a level within the physical hierarchical structure of the file (BLOCK is a record type). Within the record for this block instance the block can be identified as being part of a specific "urbanized area" despite the fact that the URBANIZED-AREA geographic area type does not hold a position within the physical structure of STF1B. Thus, there is no record type that relates to urbanized areas, but their properties can be derived by processing block area records. This is illustrated in figure 7.5.

When serving as a logical model of the contents of STF1B, then, the physical structure presents a very biased view of urbanized areas as geographic objects. In terms of data usage and analysis, urbanized areas are relevant entities. They have well-defined boundaries, and they are identified on metropolitan and census maps. The areas hold unique names and key values as do SMSAs or any meaningful statistical zones. Still, as logical objects of the file structure, their status is reduced to that of an attribute. They are thus subordinated to all other nesting areas of the file.

In terms of the relationships among attribute area objects and nesting area objects, the hierarchical file structure provides an ambiguous model. The geographic relationship (or lack thereof) between a state and an SMSA is as meaningful a concept as is the relationship between a state and an urbanized area. In the model of the file hierarchy, however, these associations are depicted quite differently. In fact, using this model, an urbanized area object is unrelated to all other geographic areas because it is not a geographic area itself -- it is an attribute of a geographic area.



GEOGRAPHIC
AREAS

BLOCK ID	URBANIZED AREA?	DATA
ABC	123	xxxx
DEF	123	xxxx
GHI	NONE	xxxx

BLOCK
RECORDS

FIGURE 7.5

RECORD REPRESENTATION OF
"ATTRIBUTE AREAS"

Recapitulation. In evaluating the performance of the physical hierarchical structure of census files we see that it does a poor job of providing the data user with a logical model of the contents of the data set. Although the structure may meet the requirements of efficient data processing, it does not match reality in terms of interrelating the primitive logical objects of a data model. These primitives (entities, relationships, properties, values) are often misrepresented or not represented at all in the model put forth by the physical structure.

The record types of the physical file structure do not necessarily represent meaningful (to the data user) logical entity types. The user must work with geographic objects which are defined as a function of the chosen physical data structure. A reordering of the hierarchical levels of the physical file structure would demand a completely different (and equally illogical) dissection of geography.

This problem rises primarily from the fact that geography is not hierarchical in nature. If there were a natural geographic hierarchy, then a hierarchical file could be designed such that each object presented to the user (data record) depicted a logically meaningful geographic entity. Unfortunately, such is not the case.

Let us consider, for the moment, what would be the consequences if the definition of geographic partitioning did adhere to a natural hierarchical ordering. We suggest that, even in this situation, the physical hierarchical structure of census files would fail as an adequate conceptual schema. True, the physical structure would present a straight forward documentation of the contents of the data set, but it would not describe the meaning of the data; it would not capture the semantics of geography.

This "absence of meaning" is due to the fact that a semantic model of data does not deal with entities only. A major portion of the meaning of the data set is inherent in the ways that the entities are related. Unfortunately, the representation of these relationships is hardly straight forward: "If the function of a database were merely to store data, its organization would be simple. Most of the complexities arise from the fact that it must also show the relationships among the various items of data that are stored" [MART76, p. 74]. As noted above, the hierarchical file structure not only fails to represent the meaningful relationships among geographic objects, but it implies a number of erroneous associations among the involved areas.

Finally, it should be remembered that, from the logical structural viewpoint, the existing physical file structure represents some logical entities as if they are properties. If the record types of the structure are thought of as relating to the entities of the model, then the "attribute areas" contained in the file do not represent entities. Rather, they depict properties of other entities.

A consequence of this representation is that, in the logic presented by the data model, geographic objects which happen to be stored as attribute areas can have no properties. Properties are asserted about entities; not about other properties. Similarly, attribute areas hold no relationships with other geographic areas; only entities are joined in relationships [2]. These restrictions occur despite the fact that, in reality, the objects represented by attribute areas are of the same nature as those represented by the "entities" of the model.

The issues discussed in the preceeding paragraphs describe the ways in which the physical file structure misrepresents the "reality of geography." A related issue is that there does not exist a straight forward means of describing and discussing this reality. Therefore, in

order to communicate the nature of the true logical association between block numbering areas and enumeration districts, for example, natural language must serve as the medium of discourse. Such an interlocution is an overly complex discussion of what could otherwise be described in terms of a few simple constructs.

The problem here is that there is no "language" of geographic semantics; the simple constructs upon which general geographic partitioning is based have not been formalized. A result of this problem is that the documentation, discussion, and description of geographic based data sets are needlessly complicated and lengthy. This affects both data technicians and data users. The issue relates not only to census data but to any geographic based data set. This problem is addressed in the next chapter.

FOOTNOTES

[1] Under the most general definition of a tree structure the parent (children) of a node may exist at any higher (lower) level. We make use of a more restrictive definition, that is the parent (children) must exist at the next higher (lower) level. The reason for this is that such a structure is held by the hierarchical files of the application being considered.

[2] Some data modeling formalisms allow for properties to hold properties or for properties to hold relationships with entities, but such is the exception rather than the rule (see chapter 4 for a discussion of this). A simple tree structured data model can hardly be considered to connote these unusual semantics.

C H A P T E R V I I I

GEOPOLITICAL STATISTICAL DATA: LOGICAL STRUCTURE

The Semantics of Geography

The above discussion makes clear the fact that the existing physical structure of a census file performs poorly when called upon to serve as a logical model of the data set. The contents of the data set are presented in a form that is inconsiderate of user needs, and the depiction of the meaning (semantics) of the data is hardly adequate. What is needed, then, is a logical framework of these files through which an understanding of both the contents and the meaning may be easily derived by the user.

Through the principle of data independence we know that such a logical structure may be vacuous of physical structural considerations: "Given the ability of data management software to separate the physical organization of data from the users' view, or 'logical organization,' the users' view ought, in theory at least, to be formulated without concern for physical representation. The users' view of data should be in whatever form is most

convenient for them, and the data management software should do the translation between this logical organization and whatever physical organization gives efficient performance" [MART76, p. 74]. The reason for promoting such physical data independence is that it allows for the creation of a logical structure which is uncluttered and straight forward for the user -- a structure which serves to simplify rather than to confuse.

In order for a data model to serve adequately as a conceptual schema of a geographic data set, it must capture the "semantics of geography." Each geographic data set involves a number of different types of geographic area. Consequently, the semantics of a geographic data file relate to this entire set of area types. To truly understand the "meaning" of geographic based information one must assimilate the "geographic system." More than the individual components, the semantics of geography include the interrelationships among the components. A conceptual schema of geographic data must therefore model this system.

A conceptual schema of a geographic data set presents a model of some aspect of real world logical structure. Like any data model it involves the use of four primitive objects: entities, relationships,

properties, and values. The latter three objects are relevant only because they give meaning to and describe the entities. Thus, entity objects are the primary focus of the model.

At this point in the discussion we will investigate the nature of these primitive objects upon which a conceptual model of geography is based. These objects will form the structure of a schema which is aimed at capturing the semantics of the underlying geographic data.

The objects of a geographic data file. With geopolitical statistical data, a geographic area is the major object of user interest. The data describe geographic areas. For the data to provide useful information it must describe a geographic area which is meaningful to the user. Data relating to an ambiguously defined geographic zone has little informational value.

In terms of data modeling, then, we see that the entities of a geographic data model are geographic areas. The entities of a data model should be meaningful to the data user. Therefore, each geographic area represented by an entity of the model should be logically significant in the user's mind; ambiguously defined areas should not relate to the entities of the model. Recall that the data

model exists purely to aid the user.

If a logical model of geographic data is to be created, some definitions are due here. A geographic entity is a well-defined geographic area which can be located on a map. The area must be identifiable by a name or by an identification number. Example geographic entities are "Amherst," and "census block 123." An area which is not an example of a geographic entity is "the portion of the SMSA that lies within the state."

Geographic entities which are defined in a similar way for a specific purpose form a geographic entity type. Each geographic entity type has a unique name which represents all geographic areas defined as that type. Example entity types are STATE, TRACT, and BLOCK. Note that in our notation the name of an entity type is shown in upper case letters.

The individual areas (entities) which are members of an entity type are called "instances" of the type. Massachusetts and California, for example, are instances of entity type STATE. No two instances of a given entity type will ever contain the same geographic point. In other words, spacial overlap within an entity type is nonexistent (see figure 8.1a).

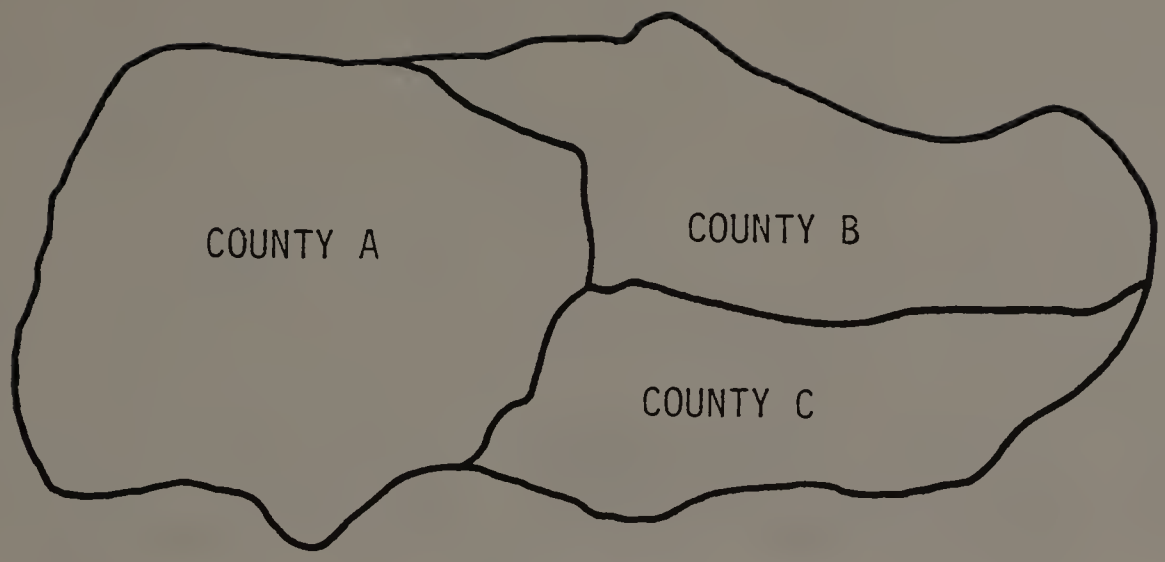


FIGURE 8.1a

INSTANCES OF AN ENTITY TYPE DO NOT OVERLAP

FIGURE 8.1b

INSTANCES OF DIFFERENT ENTITY
TYPES MAY OVERLAP

Although "intra-type" overlap is never found, "inter-type" spacial overlap is possible and prevalent. Thus, the same point(s) may be within instances of two or more entity types. Consider, for example, that each point which lies within the Town of Amherst also lies within the County of Hampshire and the State of Massachusetts (see figure 8.1b).

If the only purpose of a geographic data model were to indicate the logical contents of the data, specification of the relevant geographic entities would accomplish a major part of the task. This is not the case, however, because our data model must be the basis of a conceptual schema for the data set -- it must depict the semantics of the data set. In most applications a major portion of the meaning of the data set is captured in the relationships of the data model.

Relationships give meaning to the data model by showing the ways in which entities are associated with one another. The relationships tie together the otherwise unallied entities to form an overall "picture" of the application.

Relationships in a geographic data model have to do with the spacial overlap of the entities of the model. Overlap of geographic entities is common, and much of this

overlap is erratic and meaningless. In some cases, however, the spacial overlap of entities (or sets of entities) is well structured and meaningful. In such instances knowledge by the data user as to the nature of the overlap provides added meaning to the involved entities.

The relationships of a geographic data model, like the entities of the model, exist purely to improve the user's understanding of the data -- they should be logically meaningful objects. Consequently, the entities of the model are connected by relationships *if the overlap of the involved areas is well-defined and meaningful.* Otherwise, the entities are shown to be unrelated. In this way the semantics of the data can be clearly depicted.

Geographic relationships are of a number of different types. Each type has its own semantic and provides information about the related entities in its own way. These relationship types are defined in detail in a later section.

Regardless of type, each relationship describes the nature of the overlap (or lack thereof) that exists among geographic entities. The reader may question how a description of the spacial overlap of geographic entities

can help to clarify the semantics of the underlying data related to those entities. The answer is not straightforward; it has to do with the fact that geographic entities are often partially defined in terms of other geographic entities. This interdependence of entity specification manifests itself as well structured overlap of the involved objects. Consequently, a depiction of the structure of the overlap captures a portion of the (definitional) meaning of the objects.

As a simple example consider the previously specified definition of an SMSA: an SMSA is a set of physically contiguous counties which exhibit a high degree of economic and social integration and display certain population characteristics. From this definition of the entity it is obvious that a portion of its meaning is derived from the fact that it is made up of counties. Thus, an SMSA is a set of counties. From another perspective the definition provides an important semantic about county objects: a county may be a meaningful part of an SMSA object. This relationship is depicted in figure 8.2.

The above simple example illustrates one way that a description of the spacial overlap of geographic entities can provide meaning to the entities. Namely, entities can be combined to form other entities.

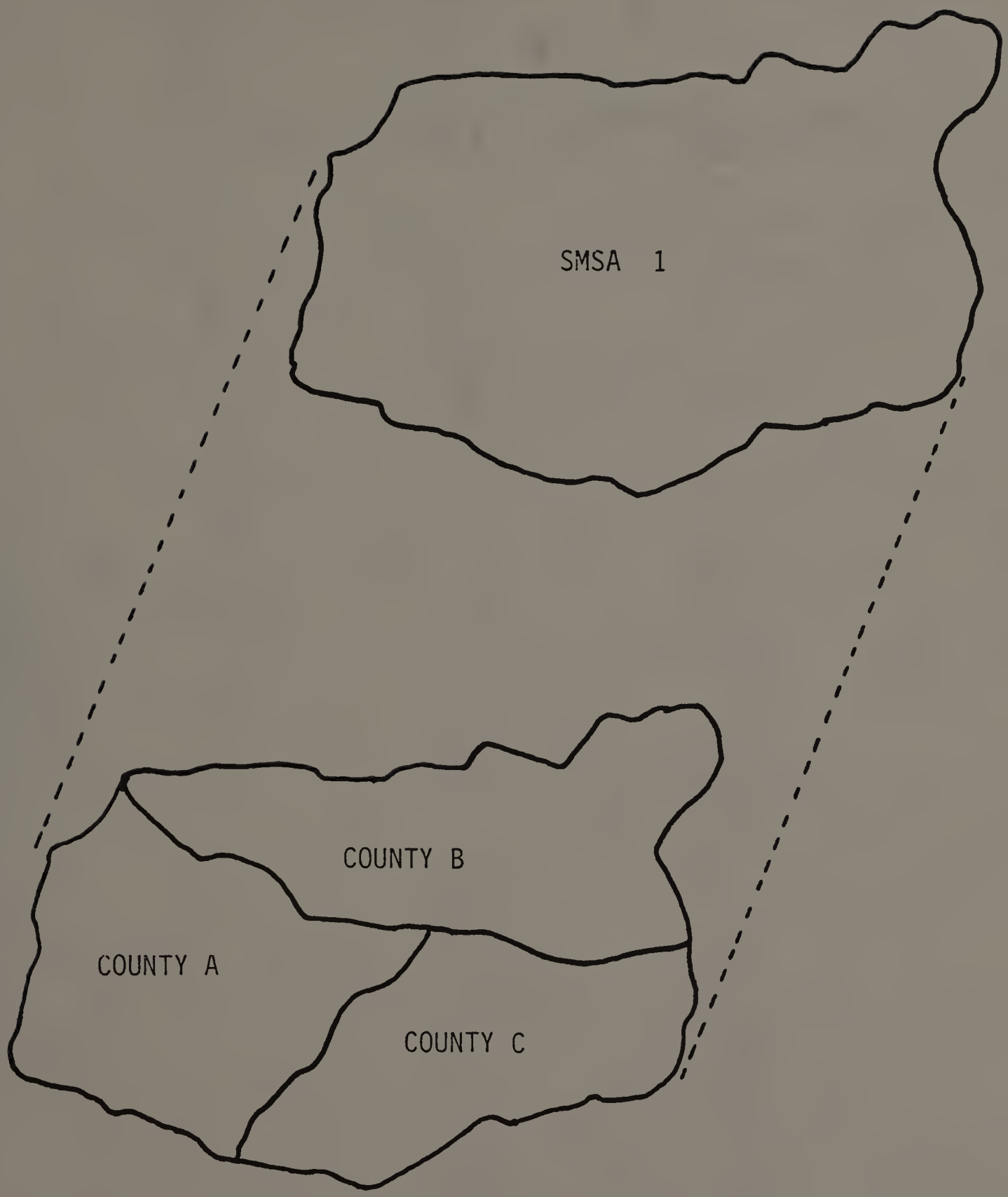


FIGURE 8.2

THE RELATIONSHIP BETWEEN
SMSAS AND COUNTIES

A second meaning-providing type of overlap occurs when two entities are restricted from sharing a common area (note that the structure of this overlap is that there is no overlap). An example of this lies in the fact that a geographic zone may be covered by an instance of a block group, or it may be covered by an instance of a tract; but it cannot be covered by an instance of both entity types. This is illustrated in figure 8.3.

The semantic of this association is that a block group is a "non-tract," and, similarly, a tract is a "non-block group." Thus, the entities are given meaning by specifying "what they are not."

Still another meaningful overlap structure exists when the overlap of two geographic entities indicates something about the overlap of other entities. For example, census tracts can lie inside or outside of SMSA entities. It happens that when a tract lies within an SMSA, the tract area is completely covered by blocks. This cannot be stated about tracts in general. Consequently, we know that if a tract instance is an "SMSA tract," then we are assured that the object comprises a set of blocks (see figure 8.4). This "meaning" is given to the tract because of its affiliation with an SMSA.

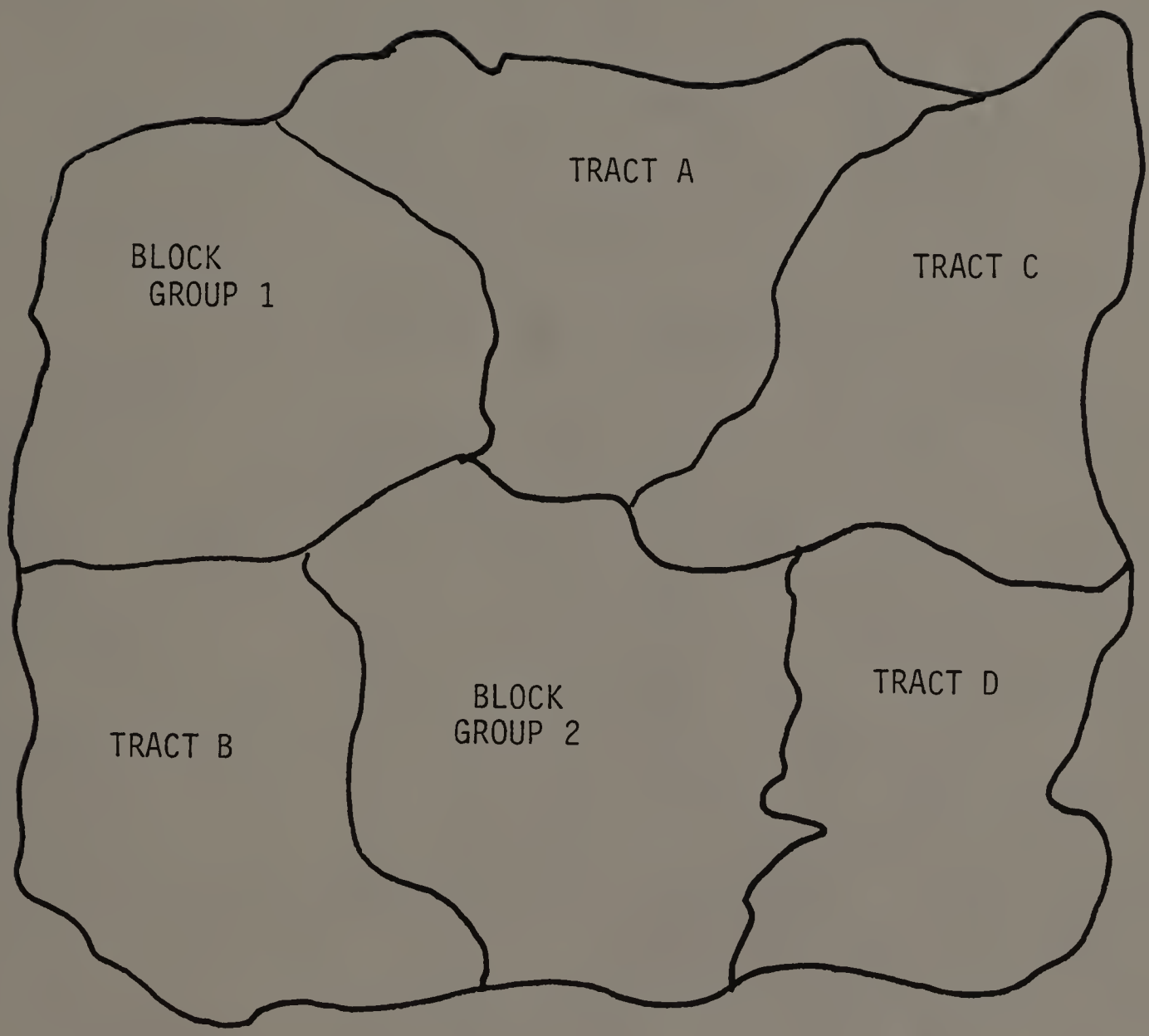


FIGURE 8.3

THE ASSOCIATION BETWEEN

BLOCK GROUPS AND TRACTS

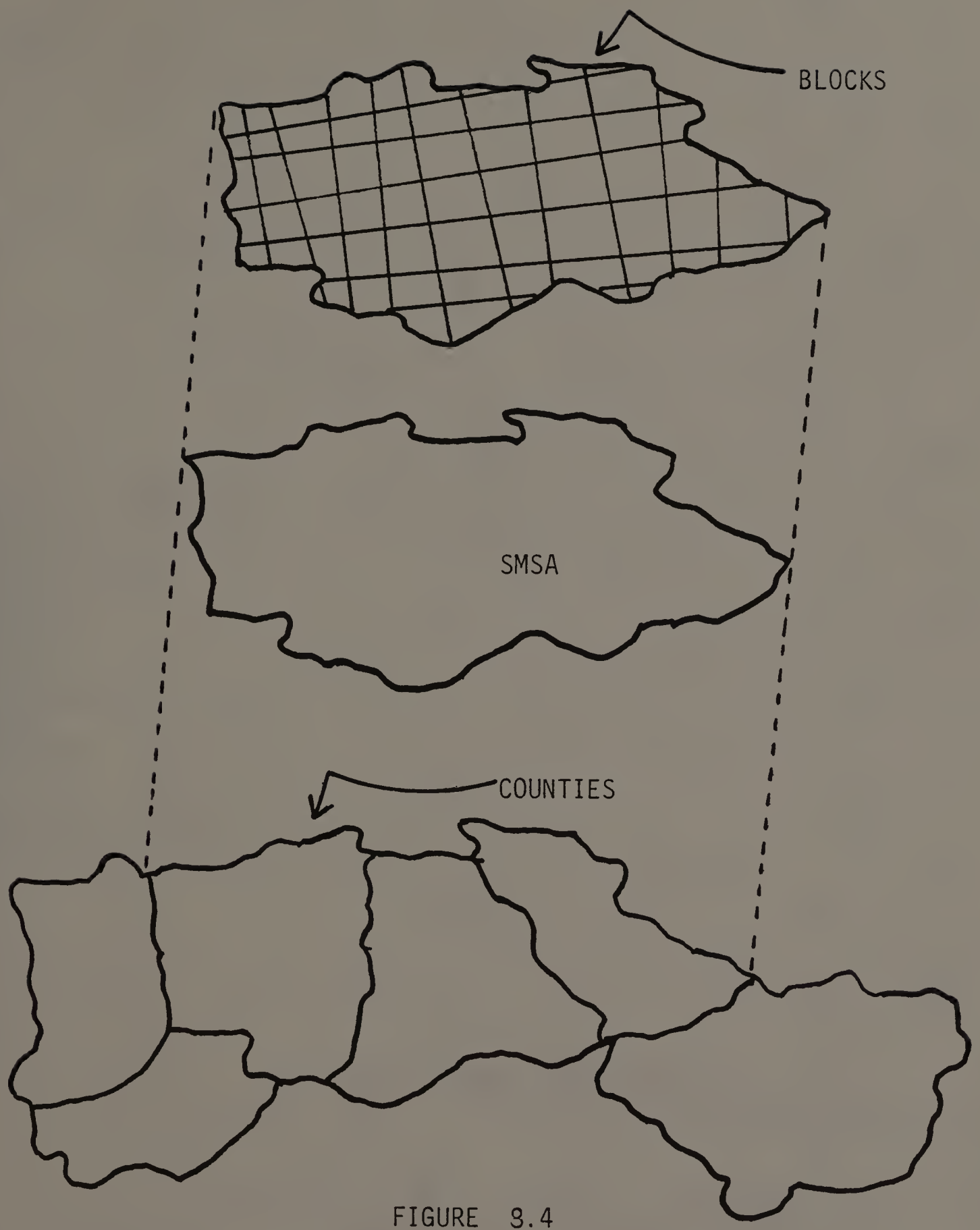


FIGURE 3.4

COUNTIES IN AN SMSA ARE

ALWAYS COVERED BY BLOCKS

The nature of the remaining two database primitives, properties and values, is not complex. Recall that the assignment of a value to a property of an entity represents an assertion about the entity. The set of all such assertions characterizes the entity.

In our geopolitical statistical application there exists a number of different entity types. At a basal level, however, each entity type depicts the same fundamental "kind" of object -- a well defined closed area of geography. Consequently, the semantic of any given property is the same for each entity to which it applies. Also, much of this semantic is portrayed in its name.

Consider, for example, the property "white population." The meaning of this property is the same whether it applies to a state or to a census enumeration district. In either case its value refers to the count of white persons living within the boundaries of the respective geographic entity [1]. Obviously, the value of the property is different for each geographic entity it relates to. The basic meaning of the property, which is apparent from its name, is constant in all cases, however. Therefore, the properties of geographic entities need be defined at most once within a total conceptual model of a geographic environment.

In most data modeling applications, value primitives are of little concern during the design of the conceptual schema. Values have to do with specific entity instances, and the schema does not deal with instances; it gives structure to entity types. In our geopolitical application, values are related to the schema structure for the following reason: values of properties of different entities are associated, and the nature of the association among the values is defined by the relationships among the entities.

Values of a geographic data set are statistical objects. Each value represents the count of the occurrence of some underlying "raw object" (e.g. person, housing unit) within the geographic entity. Geographic entities overlap, so the values of the properties of different geographic entities may be based on the same underlying raw object(s). Whenever two geographic entities share a common set of points, then, the values of a given property of the respective entities are associated.

In studying the spacial overlap of geographic entities we found that not all overlap is meaningful; only some of the entities which share a common area are linked by a relationship. From the data user's perspective,

meaningful overlap of entities should be documented and exploited. Other spacial overlap should be ignored.

Similarly, not all associations among statistical values are meaningful. When such associations are significant the semantic of the association is provided by the relationship that links the respective entities. In other words, the values of properties of different entities are associated in the same fashion as are the entities themselves. In this way the values of the data set are linked to the conceptual schema.

At this point the reader may wonder why the relation among the values of entities is an issue of importance. The answer lies in the fact that if we know the ways in which property values should be related (as defined by the relationships among entities), then we can check the integrity of the contents of the data set. Thus the structure of the conceptual schema can provide the foundation upon which semantic integrity checks are based. This is detailed in a later section.

A Geographic Data Modeling Formalism

Having introduced the basic objects of a geographic data model we are prepared to develop a data modeling formalism (DMF) through which such models can be constructed. The formalism will provide a well-defined means of structuring conceptual schemas for geographic data sets. In addition it will provide a "language" of geographic structure.

In the previous section we suggested that properties and values are only peripherally related to the semantics of geographic data. Geographic entities, and the spacial overlap structure (relationships) among these entities captures the important semantic content of a geographic data set. Consequently, the formalism is based entirely on geographic entities and geographic relationships.

The Entity-Relationship model [CHEN76; CHEN77] described in appendix A represents the semantics of a data set in terms of entities and relationships. The constructs of this formalism are not well suited to meet the specific modeling needs of a geographic data set, however. Specifically, the "meaning" of a relationship in the Entity-Relationship model does not match the semantic of spacial overlap. Also, we require a less general

depiction of the mathematical mapping (cardinality) of entities in a relationship than is provided by the Entity-Relationship model.

The DMF presented here is a revision of the Entity-Relationship model that is dedicated to meeting the specific needs of geographic based data. Each entity represented through the formalism is understood as being a geographic entity (as defined earlier). Similarly, each relationship is understood as depicting spacial overlap among geographic entities. We call this special purpose data modeling formalism the Geographic Entity-Relationship Modeling System (GERMS).

Recall that each geographic entity is, at the basal level, the same "kind" of object -- a well-defined closed geographic area. Each entity type in the GERMS therefore has the same basic graphical representation -- a rectangular node labelled with the entity type name (recall that an entity type name is spelled in capital letters).

Figure 8.5 provides some examples of GERMS entity types. Note that the GERMS, like any data modeling formalism, provides an intensional (vis-a-vis database extension) representation of objects (see chapter 4). In other words, the data model deals in entity types rather

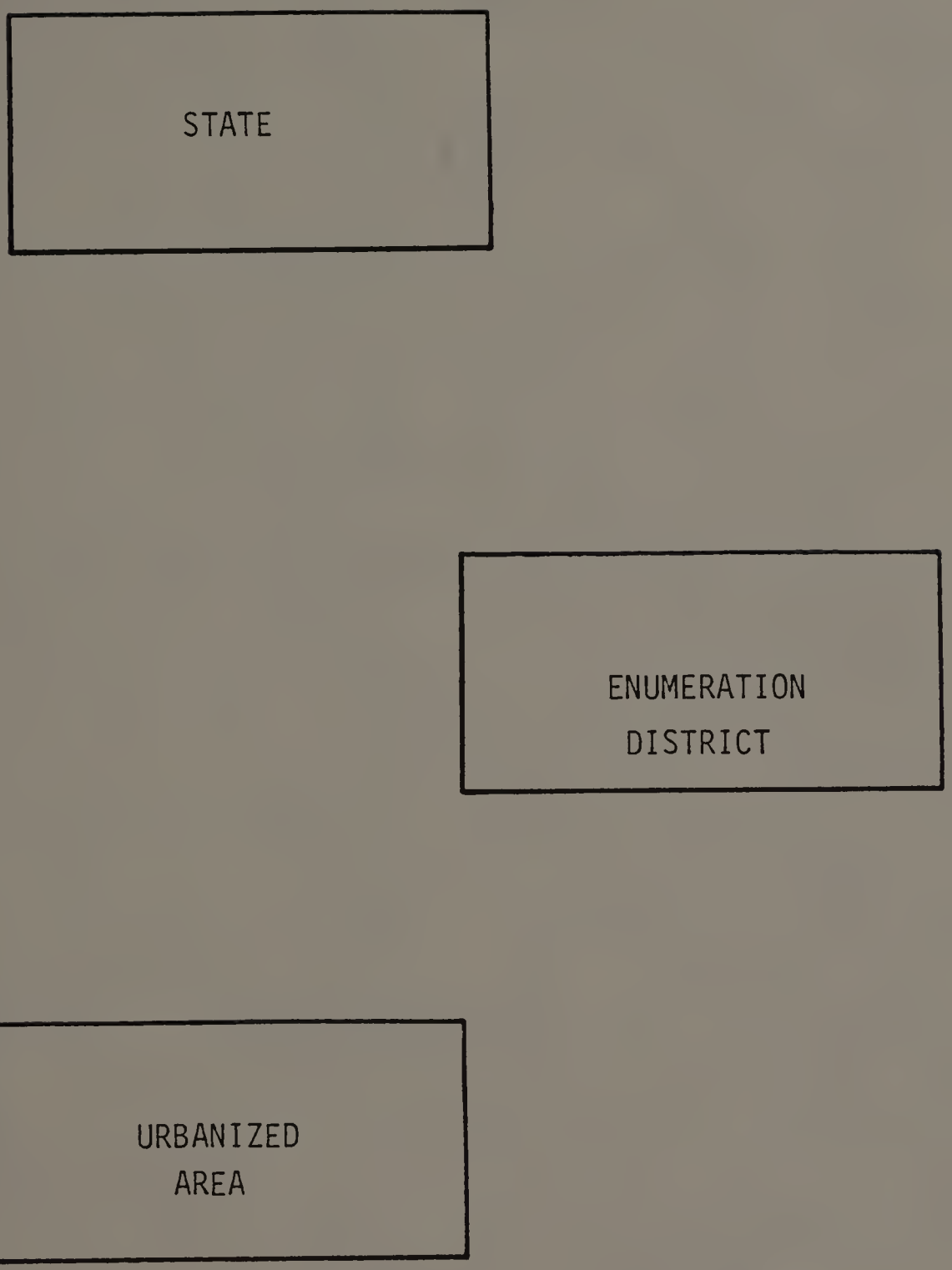


FIGURE 8.5

GEOGRAPHIC ENTITY TYPES

than entity instances. Thus, the STATE box of figure 8.5 represents Massachusetts, California, and all geographic areas which are states. Similarly, the ENUMERATION-DISTRICT box of figure 8.5 represents all enumeration districts.

At this point, before we progress any further, we would like to reiterate that an entity type is not a record type. A GERMS model serves to describe the logical structure of some data set, but this structure is (data) independent of physical structural issues. The presence of a GERMS entity type only implies that the respective geographic areas are represented in some physical form within the underlying data.

Two geographic entities may be linked through a geographic relationship. In such a case we say that a relationship "holds" between the entity types. The entity types which are joined in a relationship are referred to as "participants" in the relationship. We call the instances of the participating entity types the "constituents" of the relationship.

The existence of a relationship between two entity types defines the potential for well structured spacial overlap of instances of the entity types. The nature of the spacial overlap (if it exists) is defined by the type

of relationship linking the participating entity types. As was observed with geographic entities, there are a number of different types of relationship. Each relationship type has a type name that is shown in upper case letters.

The graphical depiction of a relationship follows the conventions of the Entity-Relationship model in that a relationship is represented by a labelled (by type name) diamond shaped node with edges (links) joining the participating entity types. Beyond this, the similarities with Entity-Relationship relationships cease to exist.

Unlike the Entity-Relationship, the GERMS graph utilizes directed edges. Each relationship has a definite "direction." The directed edge of a relationship represents this semantic.

The directed edge also portrays the mathematical mapping among instances of the related entity types. The mapping of entities in a geographic relationship is, in all cases, a functional mapping (a number of instances of one type are related to a single instance of the other type). The functionality of this mapping is given by the direction of GERMS relationship edges.

Both the GERMS and the Entity-Relationship model use the notion of relationship types [2]. Entity-Relationship relationship types are situation specific and the meaning of any relationship depends on the entities which it joins. The only pre-defined semantic of an Entity-Relationship relationship type is that it relates entity types.

GERMS relationships, on the other hand, are not situation specific; they represent pre-defined constructs [3]. In other words, the specific relationship types, which incidently are few in number, are inherent in the modeling formalism. They are, therefore, applicable to every situation.

There are a number of advantages to be gained by incorporating the definition of relationships into the constructs of the modeling formalism. The major advantage is that a single semantic is applied to many cases. Once this semantic is understood in the general sense, the relationship's name connotes an immediate understanding of its meaning in each specific application.

The reader may question how relationships, which represent the spacial overlap of heterogeneous (according to entity type) geographic areas, can be categorized into relationship types. A related question is: if it is

entity instances which overlap, how can relationships hold between entity types. The following example should help to clarify these points.

Consider drawing the boundaries of all instances of a given entity type on a sheet of cellophane film. Recall that no two instances of a given entity type can contain the same point. Therefore, none of the area boundaries on the sheet will overlap. Assume that a similar but separate sheet is created, using the same geographic scale, for each entity type.

Now, if two of these sheets are overlaid, the spacial relations that exist between instances of these entity types can be studied. In many cases the overlap of entity instances will be erratic and unstructured. Here, a GERMS relationship does not hold between the respective entity types.

In other cases the overlap will maintain a certain structure with regards to all instances of the respective types; the definition of some geographic areas is considerate of other geographic areas. In this latter situation the nature of the structure of the spacial overlap adds meaning to the involved entities. Thus a relationship does hold between the respective entity types.

If this overlaying process were carried out for all possible pairs of entity types, at least four well-defined spacial constructs could be identified. These constructs are represented by the three relationship types and the one association (non-relationship) type of the GERMS. These types are named and defined in the subsections that follow. First, however, we will note that the above illustration makes clear an important attribute of GERMS relationships. Namely, all GERMS relationships are binary -- they connect exactly two entity types.

For each relationship type described below there is provided a graphical form and a lingual form. The graphical form serves as a means through which a (logical) schema graph of a geographic data set may be created -- it is the basis of a "picture" of the semantics of geography. The lingual form acts as a standard "language" of logical geographic structure to be used in creating, discussing, and describing geographic data models. Also, the lingual form provides the basis for both a data definition language and a query language of the GERMS. The structure and use of these are discussed in later chapters.

The information content of the graphical and lingual depiction of a GERMS relationship are equivalent. That is, each form provides the same data model.

The specific relationship constructs of the GERMS are only four in number:

1. CONTAINS -- a relationship which indicates that instances of one geographic entity type lie within the boundaries of an instance of another entity type. For example, tracts lie within the boundaries of a county (COUNTY CONTAINS TRACT).

2. COMPRISES -- a relationship which indicates that instances of one entity lie within an instance of another entity type and completely cover the latter area. For example, counties lie within the boundaries of a state and completely cover the state (STATE COMPRISES COUNTY).

3. IS-A -- a relationship which indicates that instances of one entity type hold the distinction of being special case instances of another entity type. For example, some tracts (which we call urban tracts*) combine to form central business districts. Tracts, in general, do not (URBAN-TRACT* IS-A TRACT).

4. EITHER-OR -- an association (not a relationship) which indicates the grouping of a number of entity types in the formation of a single more general entity type in which the lesser distinction is lost. This is permitted only in those cases where instances of the original entity types do not overlap. For example, tracts and block numbering areas never overlap. They can therefore be grouped together to form the more general tract/bna object (TRACT/BNA IS EITHER TRACT OR BLOCK-NUMBERING-AREA).

A detailed discussion of each of these constructs in turn, together with graphical illustration of each, follows hereunder.

The CONTAINS relationship. The first and simplest type of relationship which may hold between two entity types is the "CONTAINS" relationship. When a CONTAINS relationship holds between a subordinate entity type, say "SUB," and a superordinate entity type, "SUPER," the following can be said about these types:

> Property 8.1. If a point which lies within an instance of type SUB also lies within an

instance of type SUPER, then the area defined as the instance of SUB will be completely contained within the instance of SUPER.

Property 8.1 insures that an area instance of type SUB will never be divided by the boundaries of an area of type SUPER. In alternative words, the boundaries of a SUB instance will never transcend those of a SUPER area instance.

It is important to realize that property 8.1 in no way implies a one-to-one correspondence between elements of the participating types. Rather, the mathematical mapping is 1:N from SUPER to SUB. Thus each instance of SUPER is related to (contains) any number of SUB instances, while each instance of SUB is related to (contained in) a single SUPER instance. The semantic of this relationship is illustrated in figure 8.6.

With respect to properties and values of "CONTAINS-related" entities, some conclusions can be drawn here. Recall that in this application properties are statistical objects which relate to the frequency of occurrence of some underlying "raw object" within the boundaries of the geographic area in question. When geographic areas overlap the properties of these areas share common raw objects.

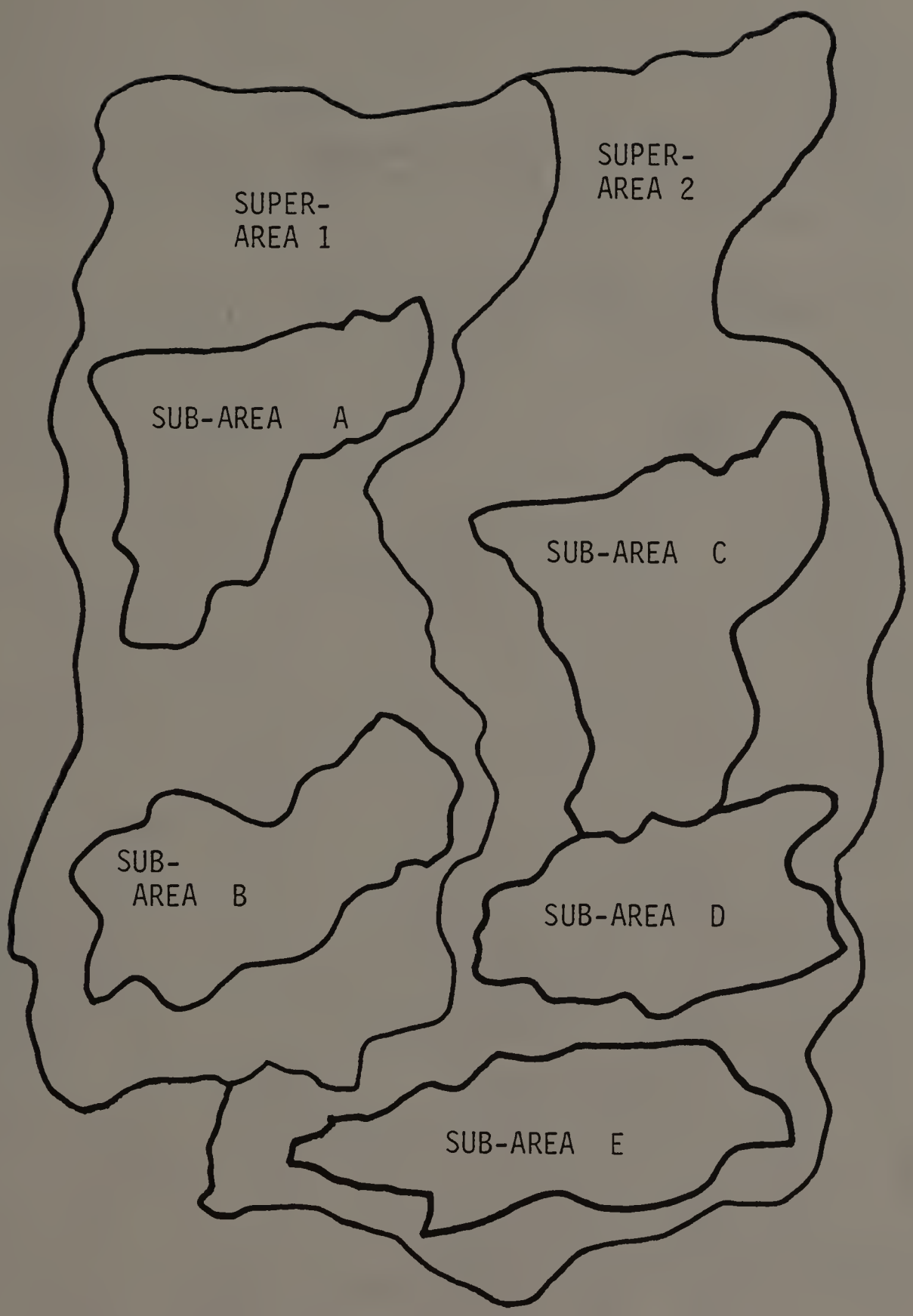


FIGURE 8.6

THE SEMANTIC OF THE CONTAINS RELATIONSHIP

"SUPER CONTAINS SUB"

When the nature of the overlap is of the CONTAINS variety, it is obvious that all raw objects which "belong" to the subordinate area also "belong" to the superordinate area. Consequently, the aggregation of the values of a given property of all SUB instances which lie within a single SUPER instance is, in all cases, less than or equal to the value of the respective property of the related SUPER instance. This fact impacts the issue of semantic integrity which is discussed later.

An example of a CONTAINS relationship is shown in figure 8.7. This graphical depiction, which in its lingual form reads "COUNTY CONTAINS TRACT," implies that a tract will never be divided by county boundaries. Note that the functional edge of the graph is directed from the subordinate (TRACT) to the superordinate (COUNTY) entity type. This directed edge represents the "direction" of the relationship (i.e. the graph does <u>not</u> indicate that TRACT COUNTAINS COUNTY).

The directed edge also represents the fact that there exists a 1:N mathematical mapping between entity instances. Therefore: (1) an instance of type COUNTY may contain many instances of type TRACT, and (2) any given instance of type TRACT will be contained in a single instance of type COUNTY.

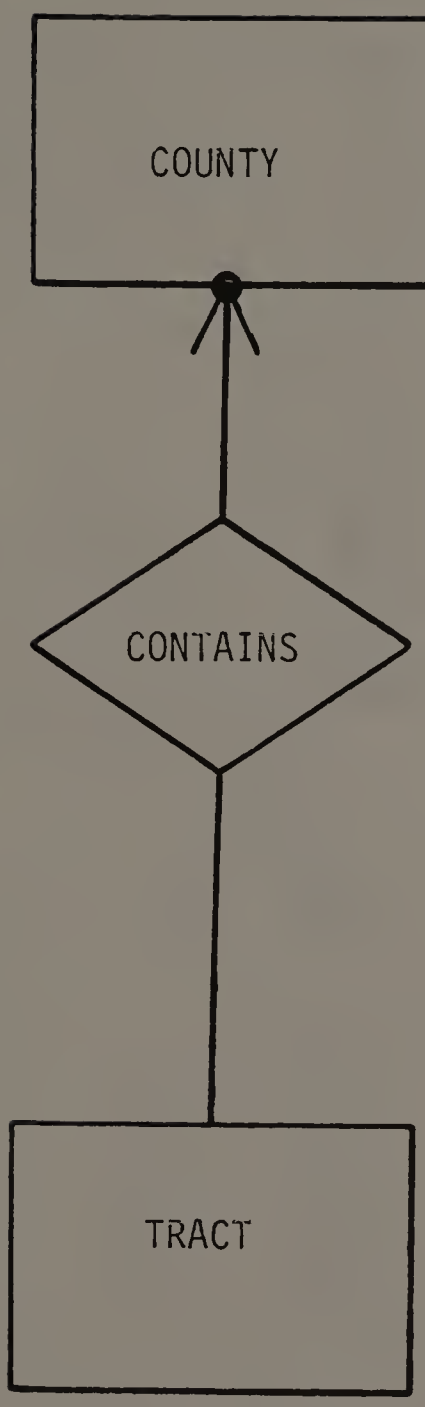


FIGURE 8.7

A CONTAINS RELATIONSHIP
"COUNTY CONTAINS TRACT"

It should be obvious that the CONTAINS relationship is transitive. Thus, if A CONTAINS B, and B CONTAINS C, then A (implicitly) CONTAINS C.

The COMPRISES relationship. The second and somewhat more powerful (in terms of meaning) relationship which may hold between two entity types is the "COMPRISES" relationship. When a COMPRISES relationship holds between a subordinate entity type, say "SUB," and a superordinate entity type, "SUPER," the following can be said about these types:

> Property 8.2.1. If a point which lies within an instance of type SUB also lies within an instance of type SUPER, then the area defined as the instance of SUB will be completely contained within the instance of SUPER.

> Property 8.2.2. The set of instances of SUB that lie within a given instance of SUPER provide complete geographic grid coverage of the instance of SUPER.

Note that property 8.2.1 denotes the fact that each COMPRISES relationship defines the existence of an implicit CONTAINS relationship between the participating

entity types. The semantic of the two relationship types is not the same, however. The distinction arises from property 8.2.1 of the latter COMPRISES relationship. This property imposes an added condition on the constituents of the relationship. Namely, the collection of subordinate areas completely cover the superordinate area in which it lies.

Figure 8.8 illustrates the semantic of the COMPRISES relationship.

An important consequence of property 8.2.1 is that it allows us to "aggregate up" from the subordinate to the superordinate entity type. In other words, all values of properties pertaining to the instances of SUB which lie within a given SUPER instance cumulatively represent the values of the respective properties of the instance of SUPER.

In the case of the CONTAINS relationship we are only allowed to make relational statements (e.g. less than or equal to) about the properties of related areas. In order to elucidate this point we will refer back to figure 8.6 (above) which illustrates the semantic of CONTAINS.

As shown in this figure, the subordinate areas of a CONTAINS relationship may cover only a portion of the respective superordinate geographic zone. This defines,

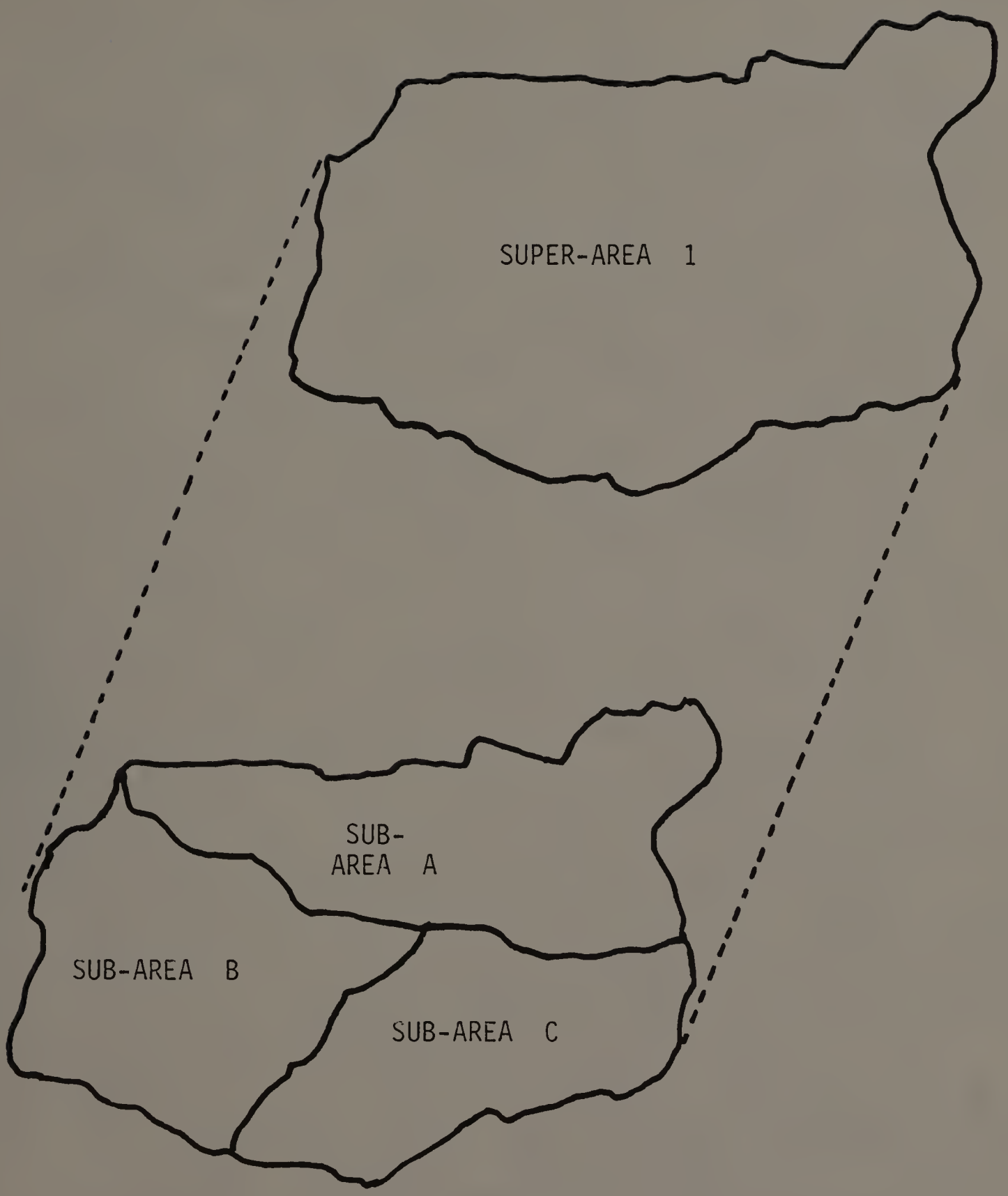


FIGURE 8.8

THE SEMANTIC OF THE COMPRISES RELATIONSHIP

"SUPER COMPRISES SUB"

because of the statistical nature of the properties under consideration, that: (1) the superordinate property values are greater than the aggregation of the subordinate property values; or (2) the superordinate property values are equal to the aggregation of the subordinate property values (this is an exceptional case). The semantic of CONTAINS only assures us that a superordinate value will not be less than the aggregation of the respective subordinate property values.

When the COMPRISES situation holds, on the other hand, we can exactly specify the values of the properties of one area in terms of the values of the properties of other areas. This is because the superordinate and the set of subordinate instances always refer to the exact same geographic area; the statistical properties must be the same. Consequently COMPRISES is called a "strong relationship" as opposed to CONTAINS which we call a "weak relationship."

Figure 8.9 has been formed by appending a COMPRISES relationship to the model in figure 8.7. Again, the directed edge from subordinate to superordinate entity type repesents both the direction of the relationship and the functional mapping (one to many) of the constituent objects.

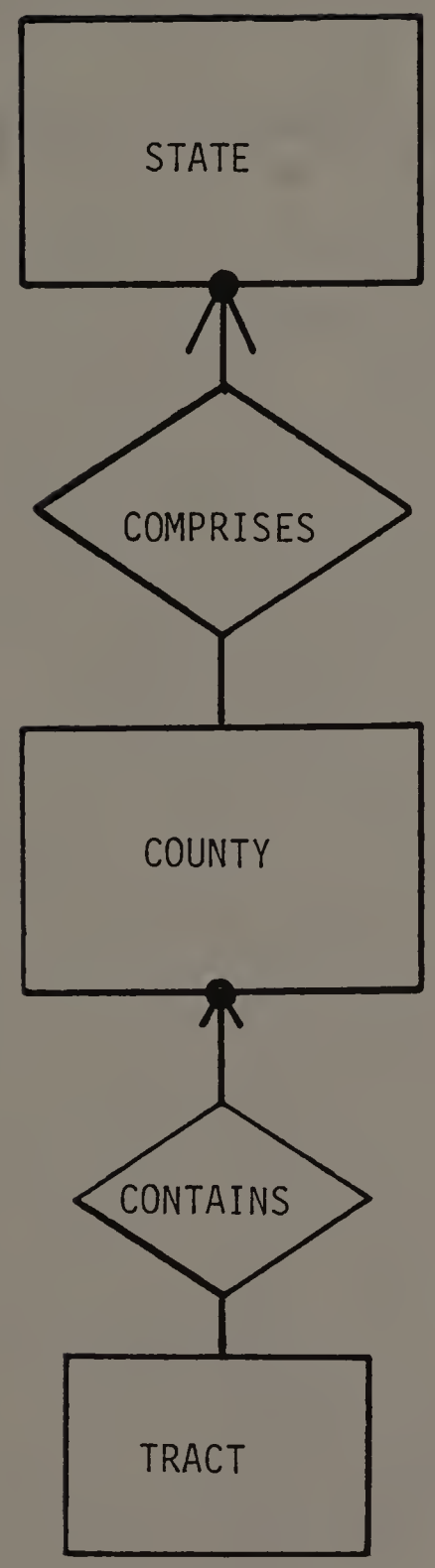


FIGURE 8.9

THE COMPRISES RELATIONSHIP

"STATE COMPRISES COUNTY"

The new relationship, read "STATE COMPRISES COUNTY," provides the following information:

1. A county will never be divided by state boundaries

2. The counties within a state completely cover the state

Like the CONTAINS relationship, the COMPRISES relationship is transitive. Therefore, if A COMPRISES B, and B COMPRISES C, then A (implicitly) COMPRISES C.

Let us again refer to the model in figure 8.9 so that we may investigate how TRACT and STATE are related. Recall that the existence of a COMPRISES relationship defines the existence of an implicit CONTAINS relationship between the participating entity types (through property 8.2.1). The model therefore provides the implicit information that STATE CONTAINS COUNTY. Furthermore, the model provides the explicit information that COUNTY CONTAINS TRACT. Because the CONTAINS relationship is transitive, we may infer that STATE CONTAINS TRACT. Thus a tract will never be divided by the boundaries of a state.

The model of figure 8.9 does not allow us to conclude that STATE COMPRISES TRACT. This is because the relationships depicted in the model do not define that tracts provide complete grid coverage of each county (although the model does not exclude this situation from the realm of possibility). Here, the presence of weak (CONTAINS) relationship inhibits the formation of a transitive "chain" of strong relationships.

The IS-A relationship. The third relationship, called "IS-A," allows for the semantic description of subcategories of an entity type. The existence of an IS-A relationship between two entity types represents the fact that one of the participating entity types is a special case of the other entity type.

When an IS-A relationship holds between two entity types, say "SPECIAL" and "GENERAL," such that type SPECIAL is a special case of type GENERAL, the following can be said about these entity types:

Property 8.3.1. Each instance of type SPECIAL is an instance of type GENERAL.

Property 8.3.2. All implicit and explicit CONTAINS, COMPRISES, and IS-A relationships which hold between type GENERAL and other entity types are inherited by type SPECIAL through the IS-A relationship (recall that each explicit COMPRISES relationship defines the existence of an implicit CONTAINS relationship).

We call property 8.3.2 the "inheritance property" of the IS-A relationship. Because of the inheritance property, entity type SPECIAL may hold its own explicit relationships as well as holding implicit relationships inherited from type GENERAL. Note that this property defines that the IS-A relationship is transitive.

In figure 8.10 the model of figure 8.9 is expanded so that we may study an example of an IS-A relationship. Here, we introduce three new entity types: CENTRAL-BUSINESS-DISTRICT, BLOCK-GROUP, and URBAN-TRACT* (pronounced urban tract-star). The asterisk appended to the name of the latter entity type is used to denote the fact that a non-standard name has been created for the purpose of naming the entity box.

From figure 8.10 it can be seen that CENTRAL-BUSINESS-DISTRICT COMPRISES URBAN-TRACT*. Also, URBAN-TRACT* COMPRISES BLOCK-GROUP. The implications of these two new relationships are straightforward. Namely:

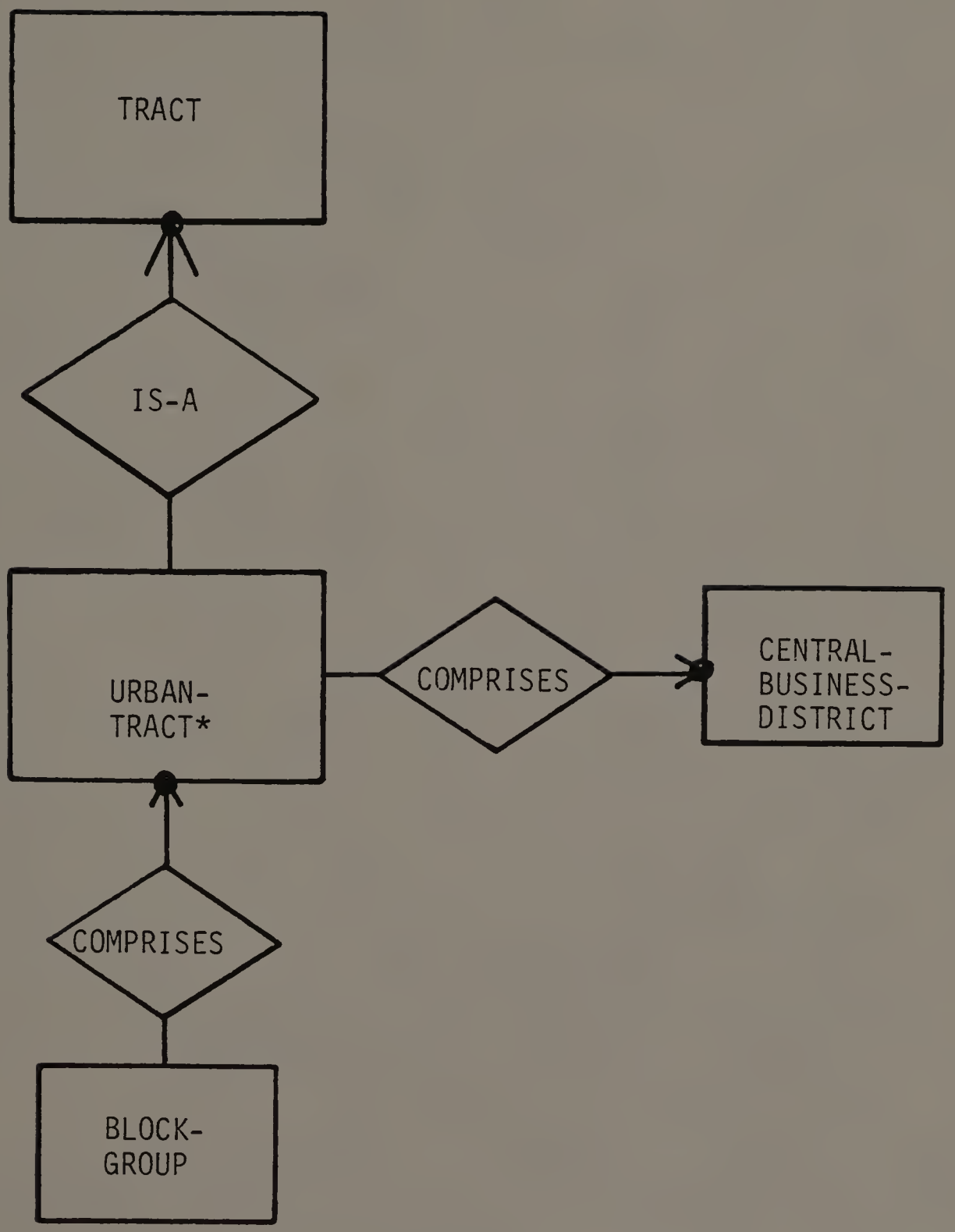


FIGURE 8.10

THE IS-A RELATIONSHIP

"URBAN-TRACT* IS-A TRACT"

1. An urban tract* will never be divided by the boundaries of a central business district, and the urban tracts* within a central business district completely cover the central business district

2. A block group will never be divided by the boundaries of an urban tract*, and the block groups within an urban tract* completely cover the urban tract*

3. A block group will never be divided by the boundaries of a central business district, and the block groups within a central business district completely cover the central business district (implied COMPRISES relationship)

The remaining new relationship, which is read "URBAN-TRACT* IS-A TRACT," defines that each instance of the type URBAN-TRACT* is also an instance of the type TRACT (see figure 8.11). From the inheritance property of the IS-A relationship it can be deduced that an urban tract* will neither cross the boundaries of a county nor those of a state.

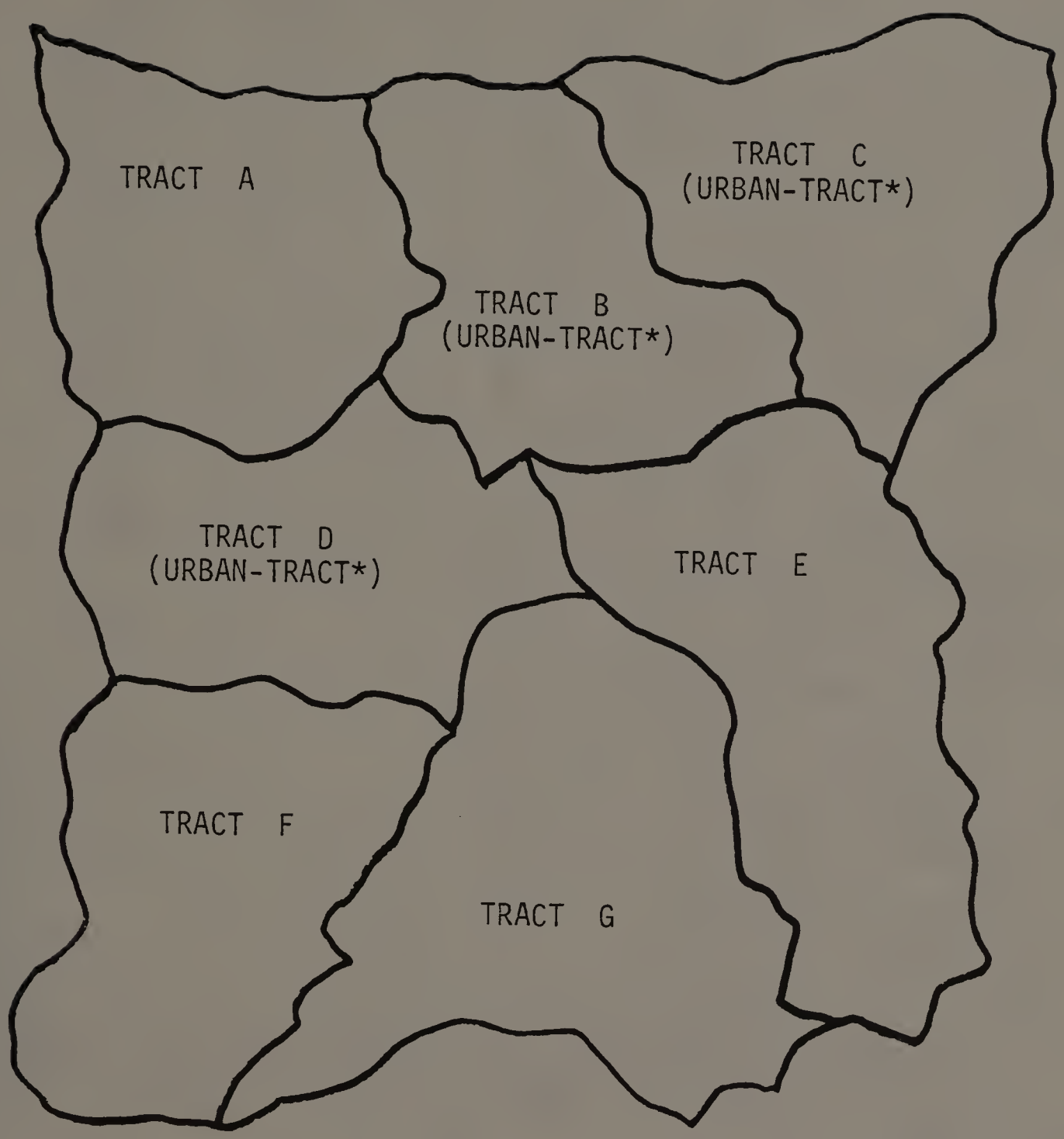


FIGURE 8.11

THE SEMANTIC OF
"URBAN-TRACT* IS-A TRACT"

The reader should note that the IS-A relationship is directed from the subtype (special case) to the supertype (general case) node of the graph. Like the edges of the previously discussed relationship types, the directed edge of the IS-A relationship represents a functional mapping between the participating entity types. Thus each instance of the special case entity type is mapped into exactly one instance of the general type.

Unlike the aforementioned relationships, the mapping of objects is not 1:N in the IS-A situation. Rather, the relationship is characterized by a 1:1 mapping. The relationship is "unbalanced," however, in that each instance of the supertype need not be linked to a colleague instance of the subtype [4]. For this reason the directed edge also serves to insure the semantic clarity of the relationship (i.e. X IS-A Y is <u>not</u> the same as Y IS-A X).

The rationale for creating the type URBAN-TRACT* as a subtype (IS-A) of TRACT lies in the fact that it allows for the presentation of more detailed information about tracts which lie within central business districts. Note that the type URBAN-TRACT* holds a direct relationship (COMPRISES) with the type BLOCK-GROUP. A COMPRISES relationship does not hold between TRACT and BLOCK-GROUP,

however (the association between tracts and block groups is discussed in the next subsection). Therefore, when it is known that a tract lies within a central business district, then it is known, without question, that the tract is completely covered by block groups. This cannot be said about tracts is general. This is illustrated in figure 8.12.

The EITHER-OR association. Geographic entity types may be linked through an "EITHER-OR association." When an EITHER-OR association holds between two entity types, say "TYPE-A" and "TYPE-B," the following can be said about these entity types:

> Property 8.4. The geographic areas defined as instances of TYPE-A and TYPE-B are mutually exclusive such that a point which lies within an instance of TYPE-A will never lie within an instance of TYPE-B.

Property 8.4 implies that an area of geography may be covered by an instance of the entity type TYPE-A or by an instance of entity type TYPE-B, but not both. Thus, as is observed with instances of a single entity type, instances of "EITHER-OR associated" entity types do not

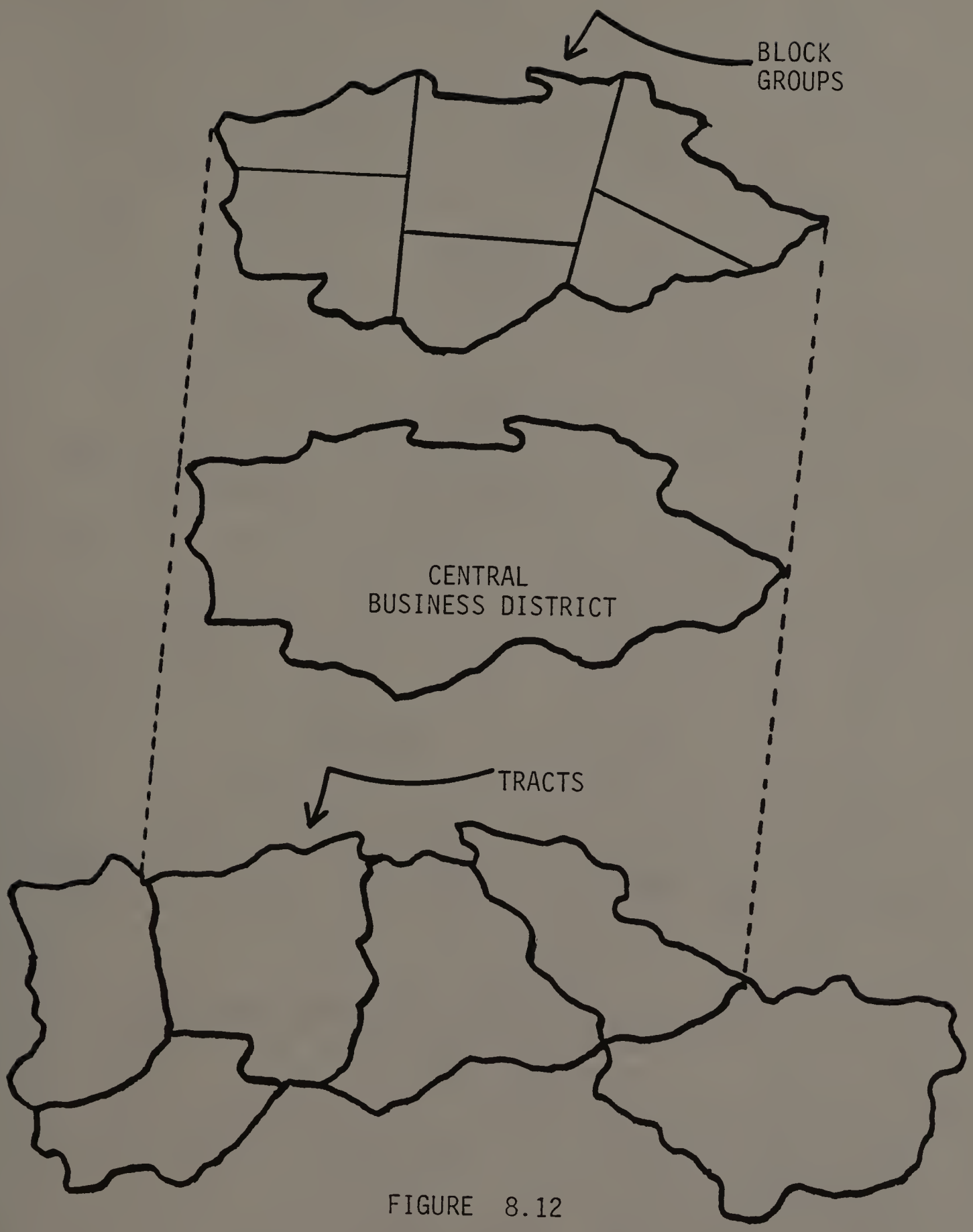


FIGURE 8.12

URBAN TRACTS*

overlap.

Because of the mutual alternation of instances of such entity types, the set TYPE-A, TYPE-B can be combined to form a single "higher level" entity type: TYPE-A/TYPE-B. From the higher level perspective all instances of the type are isomorphic -- there is no difference between a TYPE-A area instance and a TYPE-B area instance. The higher level entity type may hold its own relationship with other entity types. The EITHER-OR concept can be directly extended, if need be, to associate more than two entity types.

The EITHER-OR association <u>is not a relationship</u> between entity types. A relationship, as used in our geographic application, represents the potential for spacial overlap of instances of participating entity types. It is the nature of this overlap of constituent instances that defines the type of relationship held between the entity types. The EITHER-OR association, on the other hand, can never link instances of two entity types. This is because the association represents the fact that no two constituent instances will ever be "present" in the same geographic location. Therefore, <u>the existence of an EITHER-OR association denotes, by definition, the lack of a relationship between the participating entity types.</u>

Because the EITHER-OR association is not a relationship we do not use a the relationship (diamond) symbol in its graphical depiction. Rather, the associated entity type (rectangle) symbols are placed side by side and separated by a double bar. These juxtaposed entity boxes are surrounded by a larger rectangle.

The double bar separation of the subsidiary entity types represents the mutual exclusion of geographic coverage of these entity types. The larger peripheral entity box depicts the existence of the "higher level" entity type which is defined by the association of its subsidiary entity types. Figure 8.13a illustrates this graphical representation. The lingual form of the illustration is "TYPE-A/TYPE-B IS EITHER TYPE-A OR TYPE-B" (the ordering of the latter two entity types is irrelevant). The related semantic is shown in figure 8.13b.

The existence of an EITHER-OR association among a set of entity types designates the implicit coexistence of an IS-A relationship between each subsidiary entity type and the higher level type defined by the association. For example, let us again refer to figure 8.9 and assume that there exists an EITHER-OR association between entity type TYPE-A and entity type TYPE-B. This association defines

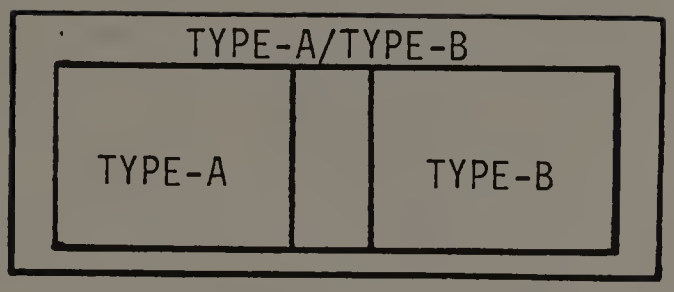


FIGURE 8.13a

THE REPRESENTATION OF
"TYPE-A/TYPE-B IS EITHER TYPE-A OR TYPE-B"

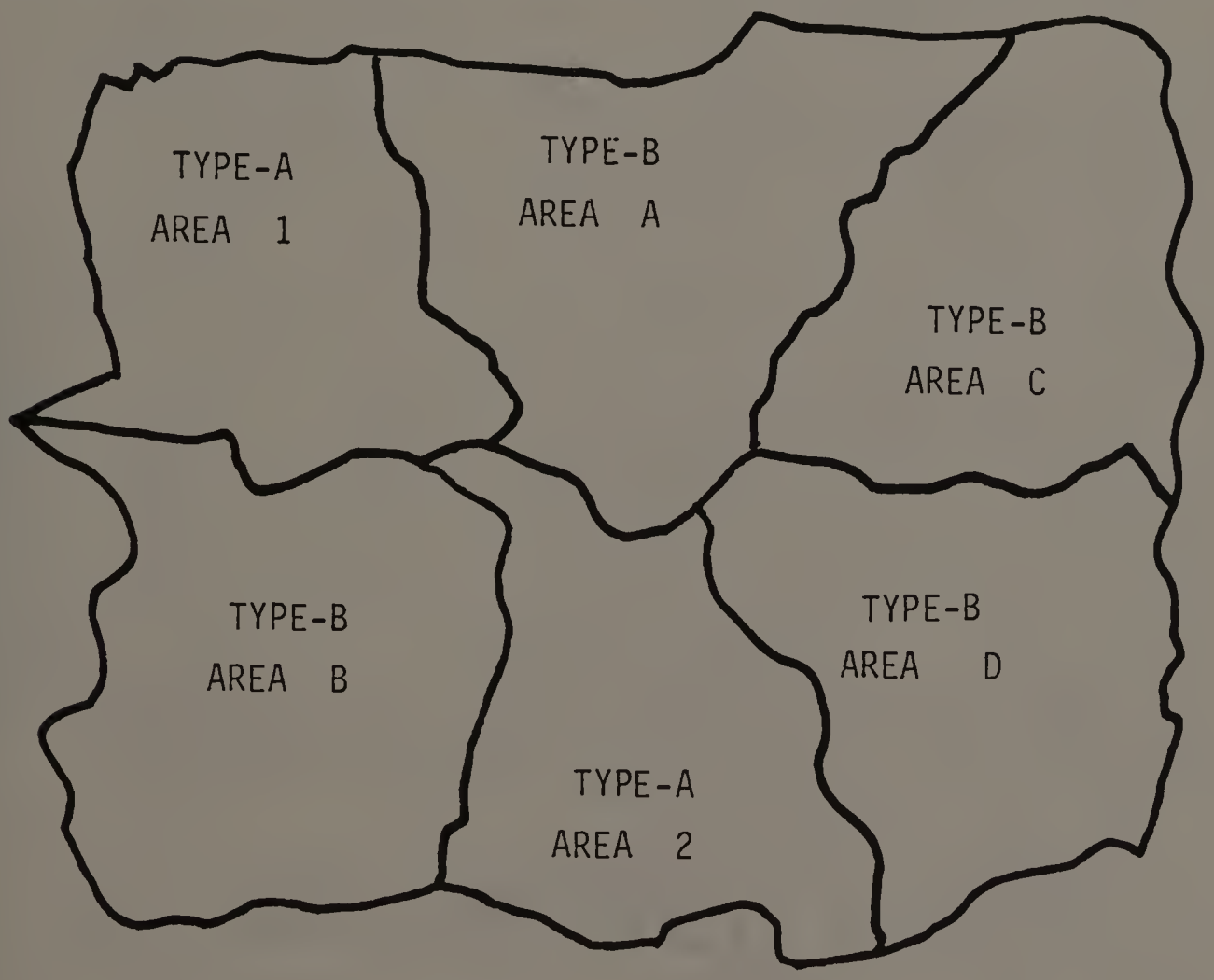


FIGURE 8.13b

THE SEMANTIC OF
"TYPE-A/TYPE-B IS EITHER TYPE-A OR TYPE-B"

the formation of the "higher level" entity type TYPE-A/TYPE-B.

Although there is no relationship between TYPE-A and TYPE-B (no overlap of respective instances), the EITHER-OR association implies the existence of an IS-A relationship between type TYPE-A (or TYPE-B) and type TYPE-A/TYPE-B. In other words, each instance of TYPE-A (TYPE-B) is an instance of TYPE-A/TYPE-B. Furthermore, because of the above noted inheritance property (property 8.3.2), all implicit and explicit CONTAINS and IS-A relationships held by entity type TYPE-A/TYPE-B are inherited by types TYPE-A and TYPE-B.

The model in figure 8.14 shows the figure 8.10 version after the inclusion of two EITHER-OR associations. We will use this model as an example of a "typical" GERMS conceptual schema of some hypothetical data set. Because a data set exists prior to the creation of the schema, the entity types which are represented in the schema is not a factor which is under our control -- we must accept the list of entity types as given.

The relationships of the schema, then, are those which relate logically the entities of the data set. Thus, a schema does not show that COUNTY COMPRISES SMSA unless the data set holds data about both counties and

SMSAs. To restate, a GERMS schema models the logical contents of a specific geographic data set. In the case here we are considering a hypothetical data set which includes six "basic" entity types: COUNTY, BLOCK-NUMBERING-AREA, TRACT, CENTRAL-BUSINESS-DISTRICT, and ENUMERATION-DISTRICT (the remaining three entity types are a manifestation of the GERMS schema creation process).

In the diagram entity type TRACT and type BLOCK-NUMBERING-AREA (BNA) are joined in an EITHER-OR association to form the higher level object TRACT/BNA. The direct implications of this association are as follows:

1. An area of land may be covered by a tract or by a block numbering area, but an area will never be covered by an instance of both types

2. Each instance of the type TRACT (and BLOCK-NUMBERING-AREA) is an instance of entity type TRACT/BNA and thus inherits the appropriate relationships with other entity types

The second EITHER-OR association of figure 8.14 denotes a similar meaning for the entity types BLOCK-GROUP and ENUMERATION-DISTRICT.

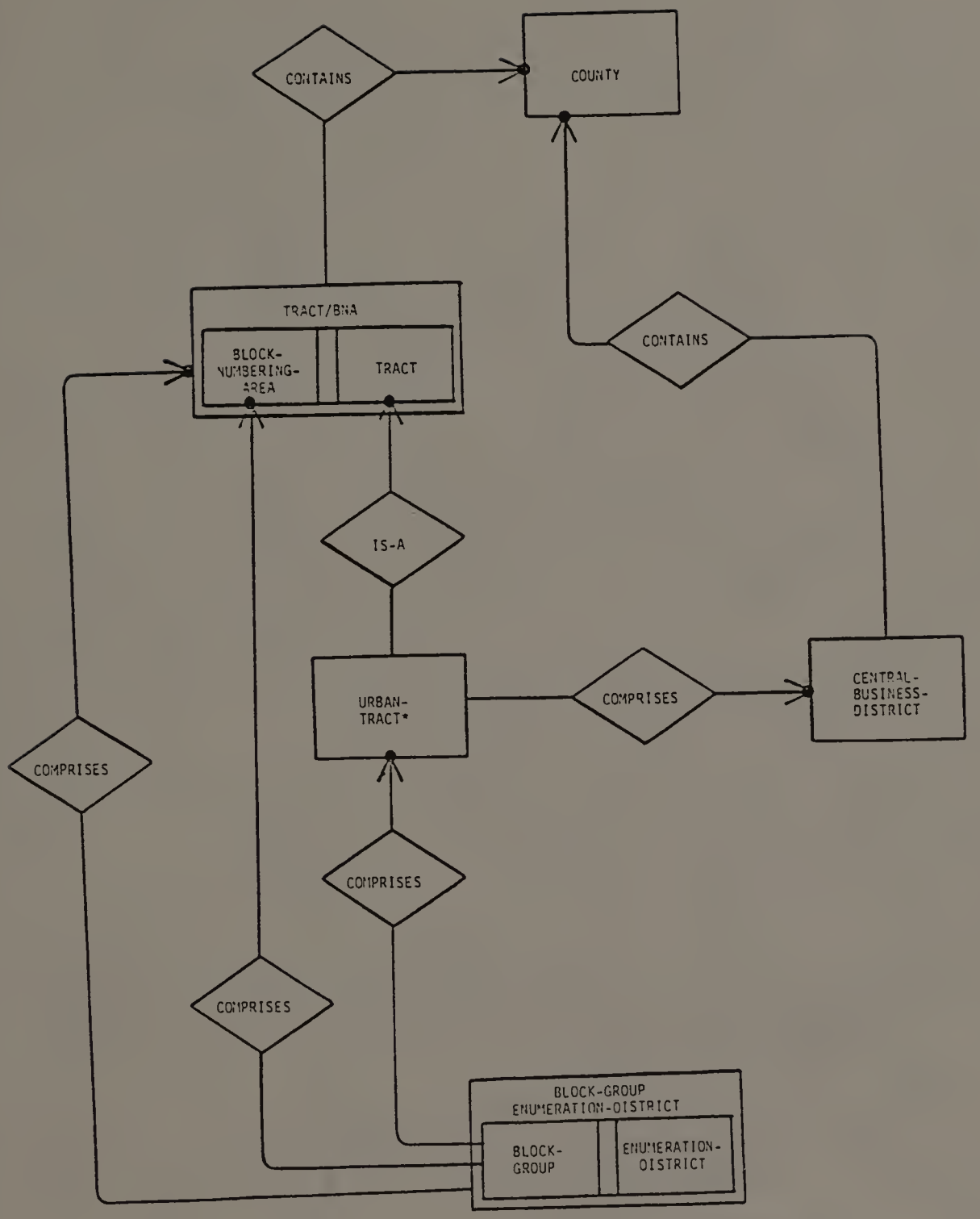

CONTAINS
COUNTY
TRACT/BNA
BLOCK-
NUMBERING-
AREA
TRACT
CONTAINS
IS-A
URBAN-
TRACT*
COMPRISES
CENTRAL-
BUSINESS-
DISTRICT
COMPRISES
COMPRISES
COMPRISES
BLOCK-GROUP
ENUMERATION-DISTRICT
BLOCK-
GROUP
ENUMERATION-
DISTRICT

FIGURE 8.14

The model shown in figure 8.14 demonstrates how relationships can be held by higher level entity types such as TRACT/BNA, or by their subsidiary types (TRACT or BLOCK-NUMBERING-AREA). Now we see that COUNTY CONTAINS TRACT/BNA. Therefore, neither the boundaries of tract nor the boundaries of a block numbering area will ever cross the boundaries of a county.

This in no way implies that instances of both types are restricted from lying within the same county for the relationship holds between COUNTY and the higher level entity type. The higher level type does not "recognize" the difference between instances of its subsidiary lower level types. Consequently, the semantic of a constraint which requires a discrimination of tracts from block numbering areas, such as the constraint in question, is not and cannot be implied by the relationship shown.

A relationship held between the two higher level entity types tells us that TRACT/BNA COMPRISES BLOCK-GROUP/ENUMERATION-DISTRICT. In other words, instances of the latter type (either block groups or enumeration districts) provide complete geographic grid coverage of the tracts or block numbering areas in which they lie.

Because BLOCK-NUMBERING-AREA COMPRISES BLOCK-GROUP, we can easily deduce that an enumeration district will never lie within a block numbering area. Through analogous reasoning we arrive at the same conclusion for urban tracts*.

TRACT IS-A TRACT/BNA (implied) and TRACT/BNA COMPRISES BLOCK-GROUP/ENUMERATION-DISTRICT. With this information we know that a tract can be exhaustively partitioned into block groups, (mutually exclusive) enumeration districts, or a combination of both types. URBAN-TRACT* IS-A TRACT so, in the absence of further information, we expect the same situation to apply to urban tracts*. The model also tells us, however, that URBAN-TRACT* COMPRISES BLOCK-GROUP. Therefore, we conclude that enumeration districts will never lie within this special type of tract.

FOOTNOTES

[1] It is obvious that the property name "white population" does not connote an exact explication of its meaning. Issues as to the precise definition of "white" and "population" are unclear. These issues are related to the operational definitions of the property and, as such, are peripheral to own concerns here. It is assumed that the operational definitions of relevant properties would be available, upon demand, to the data user.

[2] In the jargon of the Entity-Relationship model, a relationship type is called a "relationship set."

[3] A construct is a concept which is deliberately and consciously invented or adopted for a special scientific purpose [KERL73, P. 29].

[4] The mathematical mapping which is portrayed by the IS-A relationship represents an example of what is called a "total one-to-one mapping." It is also a special case of the "onto mapping" (see [TS1C82]).

C H A P T E R I X

RECONNOITERING THE GERMS FORMALISM

In the previous chapter we described the GERMS (Geographic Entity-Relationship Modeling System) -- a data modeling formalism which is dedicated to meeting the special modeling needs of geographic based statistical data. The discussion, as presented thus far, has been developed without any consideration of machine implementability [1]. We have not considered the issues relating to a "GERMS-based" database management system (DBMS). In the chapters which follow this one we will investigate such issues. First, however, we will evaluate the GERMS as described to this point.

Discussion

If a data modeling formalism (DMF) is to be evaluated, we must have at hand some criteria upon which the evaluation can be based. In [SU79] we find a list of such criteria:

1. [The DMF] should be semantically rich enough for an enterprise to model different semantic properties of data deemed necessary for the operation of the enterprise

2. [The DMF] should satisfy the requirement of data independence (i.e. it should deal only with the semantic properties of data and not with the access path structure and physical structure of data)

3. [The DMF] should contain constructs to explicitly distinguish different semantic properties so that no semantic rules will be hidden in the knowledge of the DBA [database administrator] and the users, or the application programs

4. [The DMF] should be simple yet without sacrificing the clarity of different properties of data

In studying these criteria (especially 1 and 3), we realize that there must be made a close match of application needs with DMF semantic contructs. "Although the data may have other meanings which are hidden, unknown, or even irrelevant, the meaning captured by the

data model should be adequate for the purpose required" [TSIC82, p. 6]. In simpler terms, the tool should match the task.

In reviewing the data modeling needs of the geographic based statistical data management environment we find that the application requires the use of a limited number of modeling constructs. The requirements are quite specific to the situation, however. Existing conceptual approaches (those described in appendix A) fail to meet adequately the idiosyncratic modeling needs of the geographic data management environment. Semantic constructs of these models are either underemphasized or overemphasized so that there is not a close match of ability to needs.

Because of this situation we suggest that this data management environment deserves its own special purpose modeling formalism -- a conceptual DMF which is aimed specifically at capturing and representing the semantics associated with geographic based statistical data. The GERMS is such a formalism. Each GERMS construct matches a specific well-defined semantic pertaining to geographic data. As a result, we feel that the formalism performs well when evaluated under the above criteria.

In the succeeding chapters we explore the "machine implementation" of the concepts of the GERMS methodology discussed above. This exploration results in the development of a "GERMS-based" database management system in which all user-system interaction occurs "in terms of" the GERMS conceptual schema. Thus the GERMS methodology is taken beyond the realm of database design and discussion -- it becomes a tool to aid in database usage.

Before these topics are considered, we would like to relate the GERMS modeling formalism to some of the concepts discussed in the early chapters of this work. This is the focus of the following addendum. The reader who is not concerned with these issues is urged to omit the reading of this addendum and to proceed directly to the next chapter.

Addendum: The GERMS as a "Second Generation" DMF

In chapter 2 we introduced the notion of the "generations" of data modeling formalisms. The first generation formalisms (traditional hierarchical, network, and relational) have the common feature that their constructs are mainly syntactic [SU79]. Second generation

DMFs, those which are the primary focus of our discussion, concentrate mostly on data semantics. Early on in the discussion we suggested that these formalisms be referred to as "semantic DMFs."

In earlier chapters we discussed a number of concepts which are fundamental to the study of semantic DMFs. In this section we will relate the GERMS to some of these fundamental concepts. Others will be discussed in subsequent chapters. The value of this section lies in the fact that it provides insights into the applicability of general database research and ideas to the issues surrounding our specific geographic data modeling formalism.

Data abstraction. Data abstraction (as previously discussed in chapter 5) deals with the ignorance of detail about some objects. When this detail is ignored a number of (less abstract) objects are represented as a single (more abstract) object. In data modeling we usually discuss data abstraction in terms of two distinct types. The first type, aggregation abstraction, deals with the grouping of heterogeneous component objects in the formation of a single abstract object. This relates to the "PART-OF" concept as used by the artificial

intelligence community (see chapter 5 for specific examples).

Generalization, the remaining type of abstraction, deals with the grouping of homogeneous objects in the formation of a single more abstract object. It "enables a class of individual objects to be thought of generically as a single named object" [SMIT77b, p. 107]. Thus it takes advantage of the commonality that exists among objects. This abstraction is similar to the artificial intelligence concept of "IS-A" (again see chapter 5 for examples).

Aggregation abstraction does not show itself within the constructs of the GERMS formalism. The COMPRISES relationship carries the semantic that a "whole area" is made up of a number of "part of areas," but this does not match the meaning of aggregation abstraction as it is traditionally used in data modeling.

For example, if SUPER COMPRISES SUB, then the set of areas of type SUB which lie within a given SUPER area can be thought of as being component parts of the SUPER area. These component parts are homogeneous as to type, however, and are thus represented as a single object in the GERMS model. As a result, we do not see the COMPRISES relationship as representing aggregation abstraction. The

CONTAINS relationship fails to provide an example of the traditional aggregation abstraction concept for the same reason.

The EITHER-OR association of the GERMS formalism does join heterogeneous (as to type) objects in the formation of a single higher level object type. Still, the semantic of EITHER-OR does not match that of aggregation abstraction. This is because, at the instance level, the EITHER-OR association denotes that an instance of at most one lower level type is linked to a given instance of the higher level type. Aggregation abstraction, on the other hand, denotes that an instance of each involved lower level type is linked to a higher level object instance.

As for generalization abstraction, its presence is observed within the constructs of the GERMS. Recall from chapter 5 that whenever similar object instances (called "tokens" in the jargon of generalization) are collected and represented as a single named object type, the process is called "classification." Recall also that classification represents a special case of generalization abstraction. Specifically, it is token-type generalization. With this in mind, we see that the use of a GERMS geographic entity type to represent a set of

homogeneous geographic areas is an example of generalization abstraction.

The GERMS IS-A relationship is an example of type-type generalization. Here, token objects of one type represent a less generic case of the tokens of the other type. Thus, in the GERMS relationship URBAN-TRACT* IS-A TRACT, each URBAN-TRACT* token (instance of an urban tract* area) is a less generic, or more "well-defined" representation of a TRACT token.

When discussing generalization abstraction we often speak of the "property inheritance rule" (see [CODD79]). To review briefly, this principle states that the properties of more general objects are acquired or "inherited" by their less generic counterparts. Thus, in a common business database setting, the properties of an EMPLOYEE token are inherited by the respective SALESMAN token since SALESMAN IS-A EMPLOYEE. This rule makes sense because the two tokens represent the same object; a person who is employed as a salesman. It is only the way of viewing the object that changes.

The property inheritance rule does apply to the GERMS IS-A relationship (this should not be confused with our "inheritance property" which deals with relationships). Thus the population of an URBAN-TRACT*

token is the same as that of the respective TRACT token if URBAN-TRACT* IS-A TRACT. Again, the rule holds because the tokens represent the same object -- a specific geographic area.

In summary, we find that of the two forms of traditional data abstraction used in data modeling only one presents itself in the GERMS formalism. Generalization abstraction is used in the classification process in which geographic areas are grouped into geographic entity types. Generalization abstraction also appears in the GERMS IS-A construct. Aggregation abstraction is not used in our formalism.

Logical redundancy. As discussed in chapter 6, data abstraction is closely related to the issue of data user views, which in turn is related to logical redundancy. A database schema is logically redundant if some data values represented in the model are derivable algorithmically from other data values. It is important to realize that, despite this reference to physical data, logical redundancy is a logical concept. It in no way implies a specific physical structure.

Logical redundancy is often used in data models because it allows for the model to serve a greater variety of user needs; a single datum can be "viewed" in a number of different ways. It is obvious that a GERMS model deals heavily with logical redundancy. Every occurrence of a COMPRISES relationship represents the presence of logical redundancy since the properties of each superordinate area can be exactly specified as the aggregation of the properties of the related subordinate areas. This is due to the statistical nature of the data involved.

Every IS-A relationship produces logical redundancy since the relationship allows a single object to be viewed in "two different lights." Thus, an area can be viewed as being an instance of the "special case" entity type (e.g. URBAN-TRACT*), or it can be viewed as being an instance of the "general case" entity type (TRACT). In either situation the respective data values can be related algorithmically -- they are equal.

The same can be said about the EITHER-OR association of the GERMS. Here, a single geographic area can be viewed as being an instance of the "higher level" type (e.g. TRACT/BNA), or it can be viewed as being an instance of a "lower level" entity type (TRACT or BLOCK-NUMBERING-AREA). In either case the respective

properties have equal values since each representation depicts the same underlying geographic object.

In a later chapter we will investigate how a geographic data set whose logical structure is redundant can best be represented physically. As a close to this chapter we would like to reiterate that a GERMS representation is a logically redundant structure. The removal of this redundancy would greatly limit the variety of views of the data and, consequently, would complicate and limit the use of the data set.

FOOTNOTES

[1] Recall that it is only recently that the Entity-Relationship model has been used as other than a database design and documentation tool (see appendix A).

PART III

A GEOGRAPHIC INFORMATION SYSTEM

"Presumably man's spirit should be elevated if he can better review his shady past and analyze more completely and objectively his present problems. He has built a civilization so complex that he needs to mechanize his records more fully if he is to push his experiment in its proper paths and not become bogged down when partway home by having overtaxed his limited memory. His excursions may be more enjoyable if he can reacquire the privilege of forgetting the manifold things he does not need to have immediately at hand, with some assurance that he can find them again if they prove important."

Vannevar Bush
"Memex Revisited"

C H A P T E R X

CONSIDERING USER NEEDS -- QUERIES

Introduction

In an earlier chapter we presented a data modeling formalism (DMF) which is dedicated to meeting the specific needs of the geopolitical statistical data management environment. We call this formalism the GERMS (Geographic Entity-Relationship Modeling System). The constructs of the GERMS are applied in the formation of a data model of a geographic based statistical data set. A GERMS model is a semantic data model because it captures the meaning (semantics) of the underlying data.

At the present stage of our discussion the GERMS exists in a "non-mechanized," or "non-machine implemented" form. As such, the formalism provides us with two basic products:

1. A documentation medium. A GERMS model, in either its graphical or its lingual form, provides concise well-defined documentation of the logical structure of a geographic data set. The model illustrates the logical

contents of the data set as well as the meaning of said contents. This information is often derivable from alternative forms of documentation, but such derivation is needlessly complex and tortuous.

2. A communication medium. The GERMS language provides data technicians and users with a formalized means of discussing and describing the logical structural issues pertaining to geogaphic data. GERMS phrases are based on a minimal set of constructs which we feel represents the most important aspects of geographic semantics. Once the semantic of each of these constructs is understood, a "GERMS-based" discussion is concise and unambiguous. The alternative discussion medium, natural language, is hardly well suited for this purpose.

Since a GERMS data model represents the logical framework of a specific data set it serves as a conceptual schema of the data set. In its present form (that presented thus far) the schema structure, considered in conjunction with the physical structure of the data set, can help to answer two questions for the user. Specifically, the structures provide insights into answering: (1) Does the data set contain (in whatever

physical form) the information that I need? and (2) Given the geographic objects that I am interested in, where can I expect data retrieval problems and complexities to arise?

In this and the following chapters we investigate how the concepts of the GERMS can be machine implemented. In other words, we will consider the issues surrounding the creation of a "GERMS-based" DBMS (database management system). In such a system the structural framework of the data model becomes an integrated part of the DBMS as opposed to existing as a separate form of system documentation. Thus, in a system of this type, the DBMS has a "knowledge" of the logical structure of the data model as well as a "knowledge" of the physical data structure [1].

The advantages of this design are many. Access to the data can be made "through" the conceptual schema so that all objects seen by the data user are logically meaningful. In other words, the schema acts as a "structural template" through which all objects are fitted before being presented to or received from the data user.

When the schema is used in this way the user is insulated from the complexities that surround the physical data structure. She or he need never consider whether a

geographic object is an attribute area or a nesting area; whether an area is physically represented as a whole entity or as a number of scattered parts. We believe that this is as it should be for, under the precept of data independence, the physical structure should not affect the logical use of the data.

The discussion from this point on is based on the assumption that the data set resides on some direct access storage device (e.g. disk). This is necessary because our system is designed to respond to random requests for the retrieval of data. The sequential nature of the tape storage medium fails to provide the required flexibility to meet the needs of such requests.

A Geographic Information System

When the logical structure of a data set is integrated into the framework of a DBMS the system's ability to handle a certain type of user request is greatly improved. Specifically, the system is much better suited to respond to unanticipated and spontaneous information requests (we refer to such inquiries as "ad hoc" requests). In an attempt to elucidate this point we

will first investigate the complexities that surround the processing of ad hoc information requests under the alternative traditional system.

Assume that a hierarchically organized census data set exists in some "machine processable" form (e.g. a disk file). Further assume that a GERMS schema has been created for this file and exists as (separate) documentation as to the logical file structure.

Now, consider a user who, for some unknown reason, wishes to know the white population of Hampshire County "broken down" by election precinct. Unless the software required to respond to this specific request exists a priori, the following steps must be performed:

1. The presence of the geographic entities in question is verified using the conceptual schema graph

2. The semantic of the geographic relationship held between the entity types is assimilated (i.e. COUNTY CONTAINS ELECTION-PRECINCT)

3. The representations of the entities are located in the physical structure of the file

4. Given the physical structure and corresponding logical structure, a retrieval program is written to collect and possibly reassemble the required data

5. The file is processed using this program

Depending on the nature of the physical and respective logical structures involved, the above steps can constitute an arduous task. It should be noted that since the retrieval software is written with regards to a specific question and a specific data file, its applicability to other files and retrieval problems is very limited. At best we would expect it to be usable, with modification, in retrieving different attributes of the same entity types from the same physical file.

Now consider a second scenario in which the contents of the data file exist within a "GERMS-based" DBMS which has, as an integrated part of the system, a "knowledge" of the structure of the conceptual schema. In this situation the same ad hoc retrieval problem is handled as follows:

1. The presence of the geographic entities in question is verified using the conceptual schema graph (this is the same as before)

2. The semantic of the geographic relationship held between the entity types is assimilated (this too is the same as before)

3. A query which is based on this semantic is posed to the system

4. The system responds to the query by providing the requested information in the appropriate logical form

With this second system the DBMS takes on the responsibility of accepting the user's request; locating the required data in the physical structure; retrieving the physical data; and, if required, reassembling the data to provide information about logically meaningful (whole) geographic entities. These functions, which eliminate the need for special purpose retrieval programs, constitute what will be called "query processing." Thus, the system is able to accept GERMS-based queries.

We use in the term "query" in regards to the second system because in this system the DBMS takes an "active" role in the creation of information from the underlying data. We can simply "ask our questions," or "query" the system, and the DBMS responds.

As an analogy, consider a person who wishes to know the meaning of a certain word as used in a specific context. We will call this person "Able." Two of the basic options that can be pursued by Able are (1) he can consult a dictionary, or (2) he can consult a knowledgeable person who we will refer to as "Baker."

Under alternative 1, Able must first determine the proper spelling of the word. Then he must locate the proper entry in the dictionary. Finally, he must study and understand all of the definitions of the word and determine which definition best matches the context in question. Thus, the dictionary serves as a passive "bank" of words and definitions.

Under alternative 2 Able's task is much simpler. He need merely pose the question (query) to Baker and receive a meaningful answer. Here, the knowledgeable Baker takes the active role in solving the problem. With respect to our database techniques, then, using the traditional system corresponds to the dictionary method, while using the GERMS-based DBMS is analogous to consulting the knowledgeable person who can understand and answer our questions.

Because the GERMS-based system described above is able to respond to ad hoc requests, it can be considered to serve as a geographic information system (GIS) -- a system which is suited to handle one time ad hoc requests for information regarding geographic objects [2].

Some of the issues that surround such a GIS implementation are discussed in this chapter. Specifically, we consider the structure of the queries which are posed to the system. We will see that this structure is tripartite; it comprises a focus part, a range part, and a (optional) breakdown part.

Query Structure

We have suggested that there are a number of advantages to be gained by fitting the DBMS with a "knowledge" of the GERMS logical data structure. In a system of this type all inputs and outputs (queries and responses) "pass through" the model of the conceptual schema. The schema is therefore the system's interface with the data user.

The ideas that a query (or reponse) passes through the schema and that the schema serves as a structural template should obviously not be taken in the literal sense. What is meant by these concepts is that the queries are posed "in terms of" the schema. Objects and connections of the schema can be referenced in a query with complete assurance that the references will be properly interpreted, or "understood;" query references that contradict or transcend the model of the schema will not be understood. Thus, the schema serves to shape the form of queries (responses) as a template shapes the form of objects which pass through it.

Recall that a GERMS data model is made up of two basic kinds of object: geographic entity types and geographic relationship types. Of these two kinds, entities are the primary "objects of interest" of the data model. Relationships merely describe the way in which the group of entities is structured. Part of this description is based on the "direction" of the relationship.

The entity types used in a GERMS data model are specific to the data set; the relationship types are not. Relationship types, in all cases, connect exactly two entity sets (we will ignore the peculiarities of the EITHER-OR association for the present). Note that by

moving from entity type to entity type by means of the relationship types we can define paths through the structure of the schema. Such a path will be called a "chain."

With these facts in mind we can make some suggestions as to how an appropriate query should be shaped. First, however, it should be realized that, although properties are not considered in the structure of our data model, a query is a request for attributes (values of properties) of an entity(s). Thus, if we wish to retrieve information about Hampshire County, the county entity serves to delineate the range of the request (e.g. "What are the white population and the black population of Hampshire County?"). The actual data objects retrieved relate to properties of the entity (white population, black population). A query must therefore deal with entities and properties.

Again, our data model does not deal with properties because their semantic is common for all entities. Consideration of individual properties by the data model would make the structure needlessly complex.

Query focus and query range.. A query which is posed in terms of a GERMS data model must imply two distinct specifications:

1. The attributes (of the geographic areas) that are requested. We call this the "focus" of the request.

2. The geographic areas that the attributes apply to. We call this the "range" of the request.

The focus of a query can be simply stated as a list of attribute names (e.g. WHITE-POP, BLACK-POP). In the situation here, the range of a query must define both the geographic entity type and the specific entity instance(s) to which the attributes apply [3]. This can be accomplished by stating the entity type name followed by the appropriate instance identifiers, or keys (e.g. COUNTY ABC, XYZ).

It would seem that a common request would be to retrieve all instances of a given type. Rather than listing all identifiers a reserved keyword "ALL" can denote this request (i.e. COUNTY ALL).

Note that if the design of a query is to be truly GERMS-based, any entity type name used in the schema is valid for use in the specification of the range of the query. Thus, "URBAN-TRACT* ALL," and "TRACT ALL" are both valid (but very different) query range statements.

Combining the focus statement and the range statement into a meaningful form we suggest the following query structure:

$$\text{GET attlist FOR etype } \left\{ \begin{matrix} \text{keylist} \\ \text{ALL} \end{matrix} \right\}$$

where "attlist" represents some list of attributes, "etype" represents some schema entity type name, and "keylist" represents some list of entity instance identifiers. The brackets denote that exactly one of the contained elements must be used.

A request for the white population and the black population of County ABC would be specified as

GET WHITE-POP,BLACK-POP FOR COUNTY ABC

If this information is required for all counties the query would be specified as

```
GET WHITE-POP,BLACK-POP FOR COUNTY ALL
```

Geographic breakdown. The query format which is described above makes only partial use of the structure of the conceptual schema. The geographic entity types are referenced, but the geographic relationship types are not. These relationships represent meaningful overlap of the participating entity types. By making reference to these relationships in our queries, then, we can define meaningful breakdowns of our query range.

As an example consider the simple schema graph that is shown in figure 10.1 (this is a small portion of the schema for the census file PL94). Notice that our example database includes statistical data about counties, minor civil divisions (MCD), and a variety of other types of geographic area.

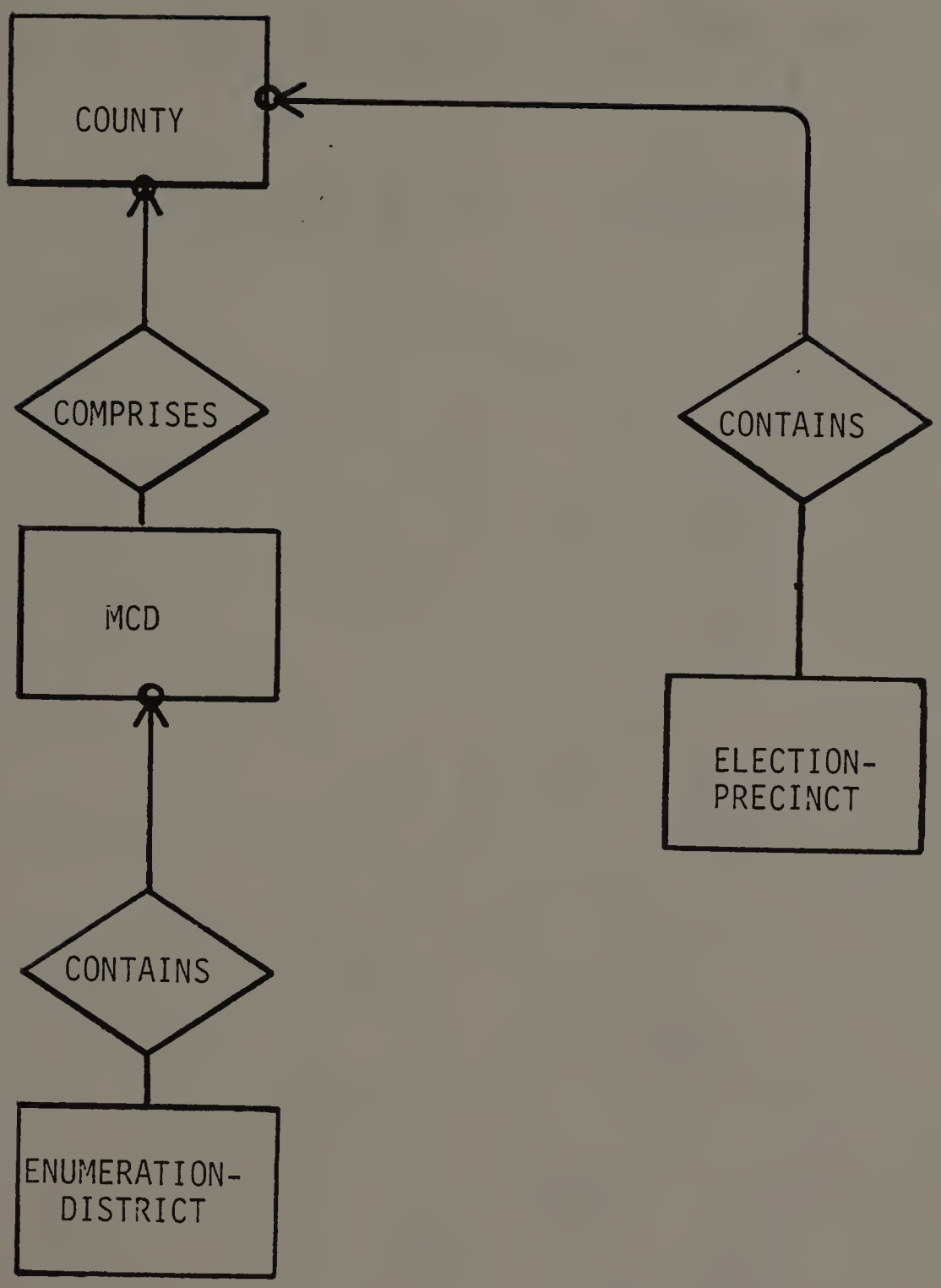


FIGURE 10.1

A SIMPLE SCHEMA

Our schema tells us that COUNTY COMPRISES MCD. Since we are assured that the overlap of counties and MCDs is meaningful, it makes sense to request that county data be presented in terms of MCD units; the respective geographic partitionings are compatible. Note that a request of this type is essentially a request for a simple report.

Even though the COMPRISES relationship that is shown in the schema graph is to be referenced in a query, we need not state that "COUNTY COMPRISES MCD" within the body of the query. This is due to the facts that (1) the DBMS has a "knowledge" of the schema, and (2) two entity types are joined by at most one relationship type within this structure. Thus, in making reference to the two entity types, COUNTY and MCD, we also imply a reference to the unique relationship that joins them.

With this in mind we suggest that data for County ABC, presented in terms of the MCDs within the county, could be requested as

```
GET attlist FOR COUNTY ABC BY MCD
```

Our database also contains data for enumeration districts. Our schema tells us that MCD CONTAINS ENUMERATION-DISTRICT, so we can request that the MCDs be further broken down thusly:

GET attlist FOR COUNTY ABC BY MCD BY ENUMERATION-DISTRICT

The above query makes use of the "chain" of schema objects which form an unbroken path from COUNTY to ENUMERATION-DISTRICT. This chain implies the existence of a relationship between COUNTY and ENUMERATION-DISTRICT. Consequently, a valid query is

GET attlist FOR COUNTY ABC BY ENUMERATION-DISTRICT

which ignores the presence of MCD areas.

Finally, note the participation of the ELECTION-PRECINCT entity type within our conceptual schema. This entity type is linked to county, but it is unrelated to all other "COUNTY-related" entities. It makes sense to request that county data be broken down by

MCDs and broken down by election precincts, but it must be made clear that these breakdowns are not to be nested.

We suggest that the keyword "AND" be used to indicate a break in the nesting of areas. Thus the query

```
GET attlist FOR COUNTY ABC BY MCD
BY ENUMERATION-DISTRICT AND BY ELECTION-PRECINCT
```

requests that county data be presented first in terms of enumeration districts within MCDs within county, and then in terms of election precincts within county.

The general format of our GERMS query is as follows:

GET attlist FOR etype1 {keylist / ALL} [BY etype2] [[AND] BY etype3]...

where the brackets enclose optional portions of the query and the ellipsis denotes that the preceeding bracketed material can be repeated.

The optional portions of the query, those that begin with the BY keyword, constitute the "breakdown" of the query. The breakdown simply serves to specify the presentation format of the data. This data relates to

those properties defined in the query focus and to those areas defined in the query range. The breakdown is essentially a simple report writer facility.

It is important to realize that the structure of the breakdown must coincide with the structure of the schema. Nesting can be requested in the breakdown if and only if the appropriate relationship (COMPRISES or CONTAINS) holds (implicitly or explicitly) in the proper direction between the entity types. Thus, the schema of figure 10.1 does not allow us to request "ELECTION-PRECINCT BY ENUMERATION-DISTRICT."

The reader should satisfy herself (himself) that the interpretation of a GERMS query by DBMS software would not be a complex task. A query in this format can be easily parsed since the ordering of the elements is strictly defined and meaningful segments are delimited by blanks and keywords. The semantic correctness of the breakdown can be verified by the DBMS software by checking the relationship structure of the schema representation which resides within the system. This "internal (to the DBMS) form representation" of the schema is discussed in the next chapter.

Discussion. The above discussion of the structure of GERMS queries is admittedly ambiguous with regards to a few issues. In this section we will attempt to clarify some of these points.

The first issue relates to the nesting of "CONTAINS related" entities as defined by the query breakdown. When the response to a query is to be nested such that the respective entity types are related via a CONTAINS relationship, the subordinate areas need not provide complete grid coverage of the respective superordinate area. This was called the "non-coverage problem" in chapter 7.

Suppose, for instance, that our query requests "COUNTY BY ELECTION-PRECINCT" when we know from the schema (figure 10.1) that COUNTY CONTAINS ELECTION-PRECINCT. It is obvious that the output "report" of the query should include "complementary data" relating to the attributes of those portions of the counties not covered by election precincts. This very tack was criticized in chapter 7 and the reader may wonder why it is suggested here. The threefold reason has to do with the nature of our GERMS based information system.

First of all, the non-coverage issue is implicit within the well-defined semantic of the CONTAINS relationship; the same cannot be said about the semantic of tree structures. Thus the non-coverage issue "makes sense" in the GERMS situation. The conceptual schema is used as a template in the formation of queries, so the issue will arise only when it is expected and understood by the user.

Second, the user need not deal with the processing of the complementary data when using our information system. The DBMS software assumes this data management task. Thus, when working with a certain type of geographic areas, the user is not forced to consider "pseudo areas" in posing a query; the query deals with true geographic objects only.

Finally, the nesting of other areas within the complementary (pseudo) areas can be avoided, if desired, by proper specification of the query breakdown. In the situation described in chapter 7 the user does not have this option because (s)he is working with a static predefined tree structure (recall this was cited as being a major cause of complexity).

Another issue which deserves clarification has to do with the impacts that the IS-A relationship and the EITHER-OR association have upon the database query process. In each of the above examples the query breakdown specification deals with COMPRISES and CONTAINS related entities only. Indeed, the use of these relationships makes sense because each of their semantics denotes a "natural nesting" relation. The same cannot be said about IS-A and EITHER-OR; their use in defining the breakdown of geographic data does not make sense. Each of these two constructs does affect the structure of queries, however.

If we consider the impacts that the IS-A relationship and the EITHER-OR association have upon the overall data model we realize that each of these constructs effects the creation of new and different entity types. The IS-A relationship is used to define a "special case" entity type, while the EITHER-OR association collapses a pair of entity types and allows them to be regarded as a single separate entity type. Each "new" entity type created through these techniques can hold its own relationships with other entity types. Note that both of these constructs essentially allow the user to "view" a single geographic area in two different ways.

Since the DBMS in our information system has full "knowledge" of the structure of the conceptual schema, any schema entity type can be legitimately referenced in a query. Therefore, the database can be queried according to any of the different views of geography presented by the model. It is only required that the structure of the query adhere to the "meaning" of the entity types as depicted in the schema. Thus the presence of IS-A and EITHER-OR in the schema allows for more flexibility in retrieving information from the database.

Summary

We have considered how the usefulness of the GERMS can extend beyond that of a documentation and communication medium. When a geographic information system (GIS) is designed to match the structure of the GERMS modeling formalism, the conceptual schema can serve as the system's interface with the user. The semantic richness of the GERMS conceptual schema provides a straightforward and natural basis for user-system interaction. Queries are posed to the GIS in a way that matches the user's understanding of the meaning of the data; system responses are provided in a similar way.

The ad hoc queries which are supported by our GIS are discussed in terms of three distinct parts: query focus, query range and query breakdown. Query focus refers to the attributes in which the system user is interested. Query range refers to the geographic area(s) to which these attributes apply. The query breakdown defines how the information requested in the focus and range is to be presented; it is a simple report writer.

In the next chapter we discuss the design of the physical database. We use this term to refer to the actual data items and the structural framework in which they exist. The logical database, which is considered in the subsequent chapter, has to do with the conceptual schema as it is represented within the system.

As initial preparation for this discussion we suggest that the reader begin to take on a "database view" of census data sets. Thus, rather than thinking in terms of individual data files, the reader should think of geographic "data pools" the contents of which may relate to any number of data files.

FOOTNOTES

[1] Obviously, we are using the term "knowledge" in a very loose sense. What is meant is that the structure is represented in some way within the computer (e.g. with pointers and data nodes), and that the DBMS accesses this structural representation through the invocation of procedures.

[2] The meaning of the term "information system," as used here, corresponds to that used in [MART75]. The architecture of a system of this type is such that it is capable of responding to spontaneous information requests. This is contrasted with an "operations system" in which the requests are known and prepared for a priori.

[3] In some database environments the entity type need not be specified in the query since an attribute may relate uniquely to a type (attribute name implies entity type name). In the GERMS case each attribute applies to all entity types. Consequently, the type name must always be included in a query.

C H A P T E R X I

THE PHYSICAL DATABASE

In this chapter we discuss the design of the physical database of our geographic information system (recall that we are no longer relying on tape as a storage medium). We are not committed to maintaining the hierarchy as a physical data structure. There are a number of reasons for this. Trees provide efficient data processing in many instances, but these efficiencies will not necessarily apply to the processing needs of our geographic information system (GIS).

Physical Database Design

If we were to invoke an ordered listing, or "dump" a census data file, an interesting fact would emerge: a census file is simply a static report. Specifically, it is a "branching report" as could be created by any standard report writer facility. The primary key of each record type in the file acts as a "breakpoint variable" in the report structure (see [ROBI80]). With this in mind, the suitability of the items of the report (data records)

to serve as the objects to be processed within our GIS becomes questionable.

The decision to strictly maintain a hierarchical physical data structure could prove to be a "straight jacket" for our application. We must not restrict opportunities for the merging and modification of geographic based data. The tree was chosen as a structure for geographic (tape) files; we are dealing with a geographic database.

The approach taken here is to design a physical database structure which is considerate of the processing needs of the GIS. These needs are obviously based on the structure of the queries which will be posed to the system, which in turn are based on the logical model of the conceptual schema. Since the physical structure is based ultimately on the logical structure, then, the mapping from the logical database to the physical database will not be complicated. For the most part, there will be a one-to-one correspondence between the objects of the logical schema and objects of the physical database.

Equivalence relations. The first step in developing our design is to identify the "equivalence relations" of the application. If the term "source structure" is taken to

mean the structure held by the data which is to be loaded into the database, then the equivalence relations equate the source structure to the structure of the conceptual schema. These relations indicate how the objects of the physical database (called the target structure) can be formed from the objects of the source structure [1]. Here, we are concerned with equivalence of information content. Thus, equivalence relations serve to locate within the source structure, the information that will form the target structure.

As an example we will use the 1980 version of the Bureau of the Census file PL-94. The hierarchical physical structure of the file, which is the source structure, is illustrated in figure 11.1. Each rectangle in this figure represents a record type. The numbers represent record type identifiers which will be used in the discussion.

Note that the records of PL-94 form a branching hierarchy (see chapter 7). Note also that there is physical redundancy of record types. Record type 3 and record type 7, for example, relate to the same kind of geographic area -- election precincts. The geographic zones represented by the instances of these record types are not equivalent in all cases, however. The type 7

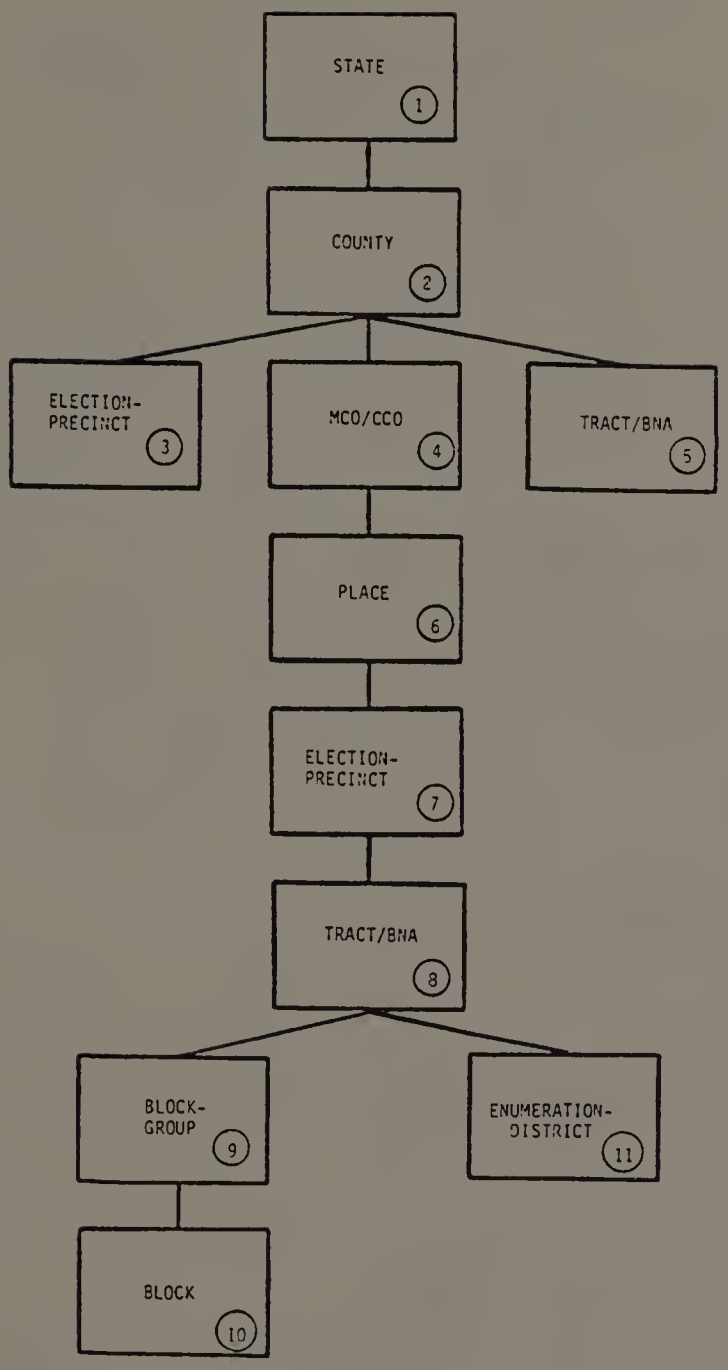


FIGURE 11.1

THE PL-94 PHYSICAL FILE STRUCTURE

records are "deeper nested" than are their type 3 counterparts and, as such, the respective geographic areas may be further dissected. Redundant representation of geographic areas is not common and its use in this particular file should not be taken to imply the norm.

Figure 11.2 provides the graphical representation of the GERMS data model of the logical contents of PL-94. The reader should notice that the GERMS model does not represent geographic entity types redundantly. The repetitive nature of the census record types is a physical structural issue and it should in no way affect the logical data structure.

Now, since each query is to be based on the geographic entity-geographic relationship framework of the GERMS schema, we should design a (target) physical data structure which matches the schema structure so as to allow the queries to be easily processed.

In processing the breakdown specification of a query we always move "down" from superordinate to subordinate entity type. In other words, we first consider a "containing" area, then we consider the "contained" areas. Each of these "downward" movements relates, with respect to the objects of the data model, to moving from a specific entity type to a specific relationship type, to a different entity type.

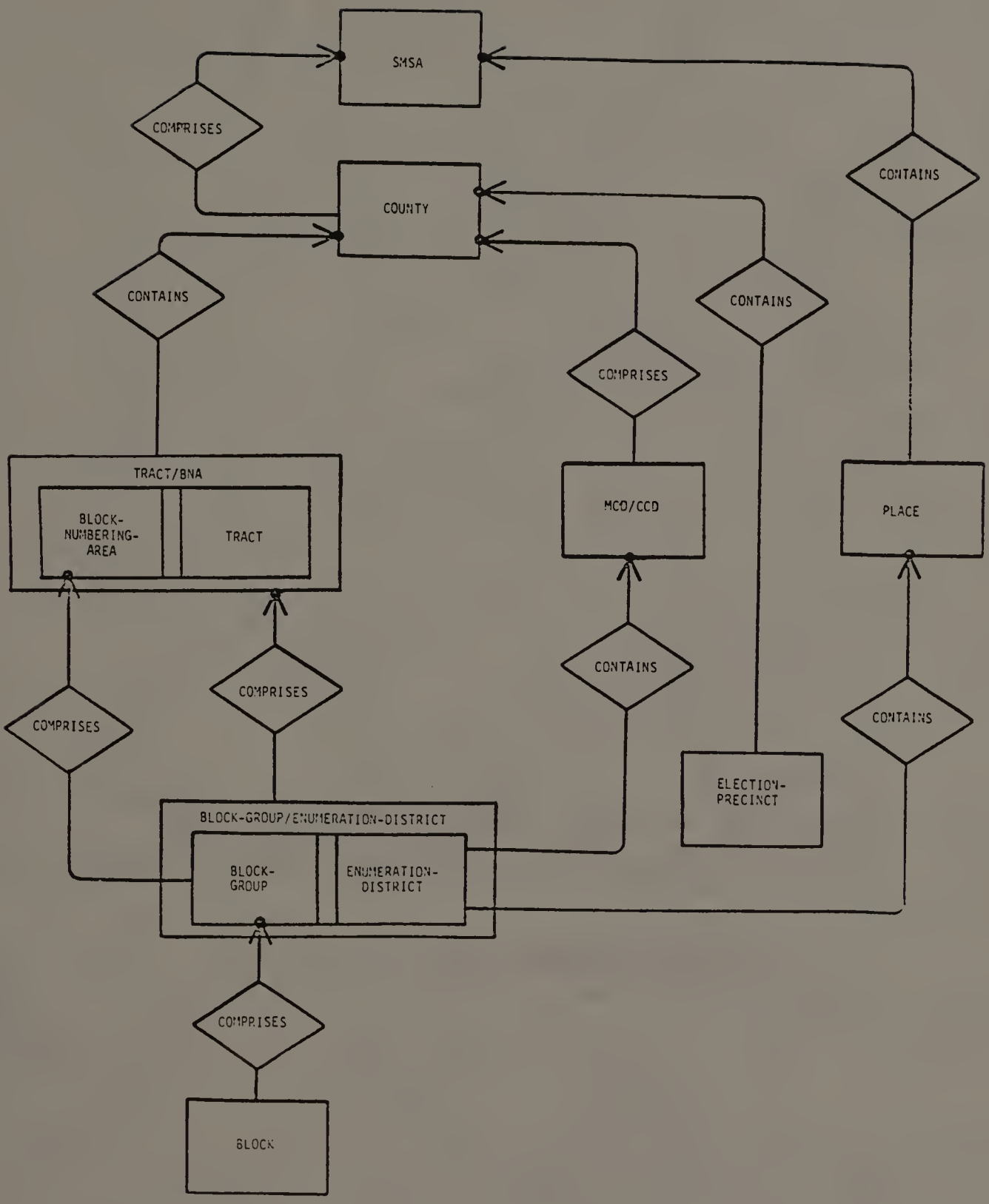


FIGURE 11.2

THE PL-94 SCHEMA

In terms of physical data structure, then, we must have some "link" from the data records that represent each superordinate object to all of the data records that represent the respective subordinate objects. Furthermore, this link must support "downward" processing.

Recall from chapter 7 that each record of the source structure (the PL-94 file) contains the key of its ancestry, but it does not contain the keys of its children records. In considering the information content of the records of the source structure, then, we see that each source record type relates to a subordinate entity type and the respective relationship of the schema structure. This is the "equivalence relation" which was mentioned above.

Figure 11.3 illustrates an instance of an equivalence relation. Here the information content of a type 3 PL-94 record (ELECTION-PRECINCT) is related to the CONTAINS relationship and to the ELECTION-PRECINCT entity type of COUNTY CONTAINS ELECTION-PRECINCT.

Again, the GERMS relationship is represented within the contents of the subordinate record type (ELECTION-PRECINCT) of the source structure because it is these records which contain the information required to form a link with the superordinate record. This link is

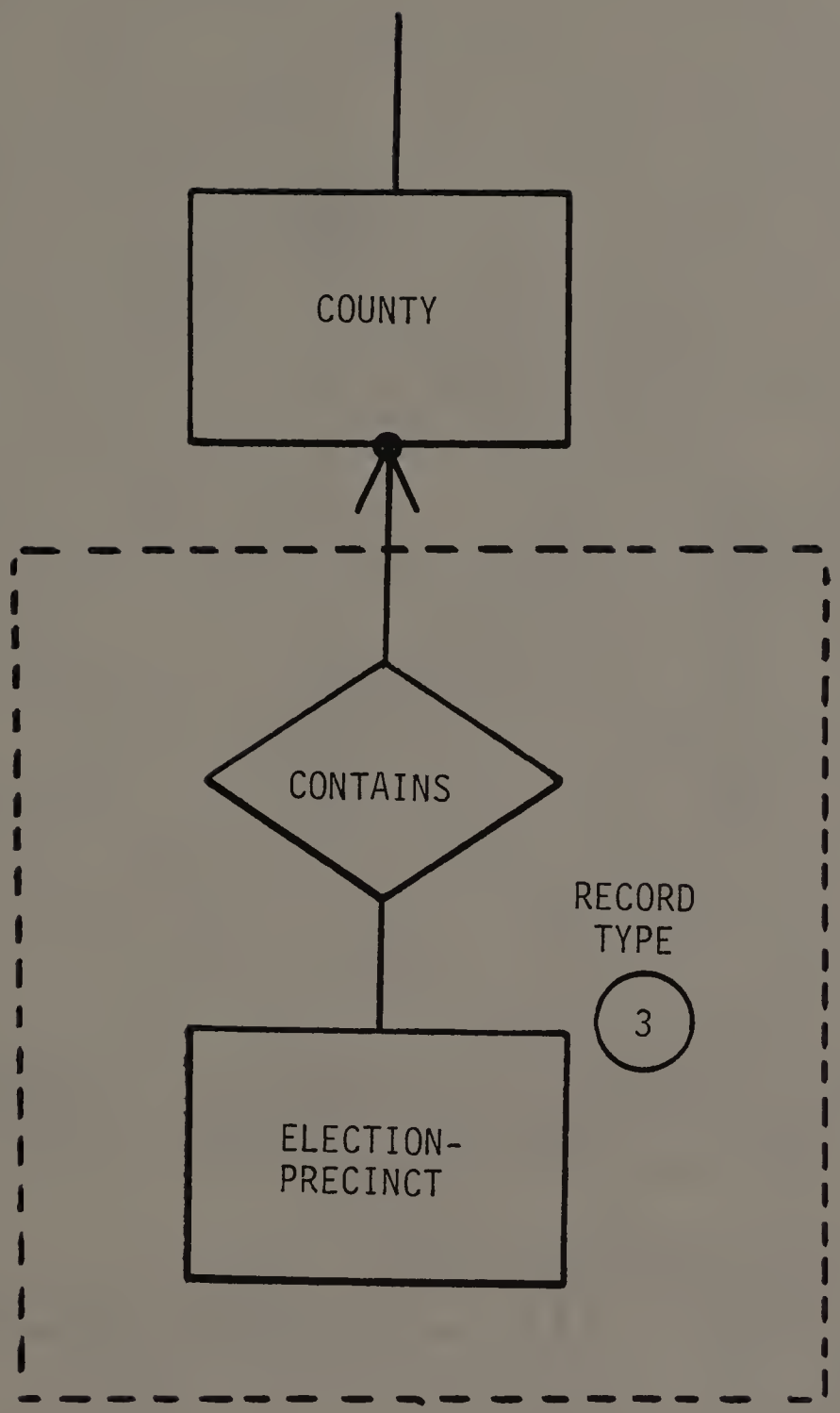


FIGURE 11.3

A PL-94
EQUIVALENCE RELATION

derivable from the key of the ancestry. Thus each type 3 record holds the information required to identify the county which it is contained in. Note that, in this special case of election precincts, record type 7 could also be used in the equivalence relation.

When designing a physical database, equivalence relations should be identified so that each relationship and entity type of the conceptual schema is accounted for. In performing this process we locate within the source structure, the information required to form the records of our target physical structure. In most instances the identification of these equivalence relations is straightforward. As would be expected, however, some cases are more complicated than are others.

Figure 11.4 illustrates one such case that arises in the PL-94 schema. Here we are dealing with two complex entity types, TRACT/BNA and BLOCK-GROUP/ ENUMERATION-DISTRICT, which are formed by two EITHER-OR associations. Both the "higher level" and the "lower level" entity types can be referenced in a query, so we must specify the equivalence relations accordingly.

Record type 5 represents both tracts and block numbering areas. Consequently, the information required to link the county entity type with type TRACT,

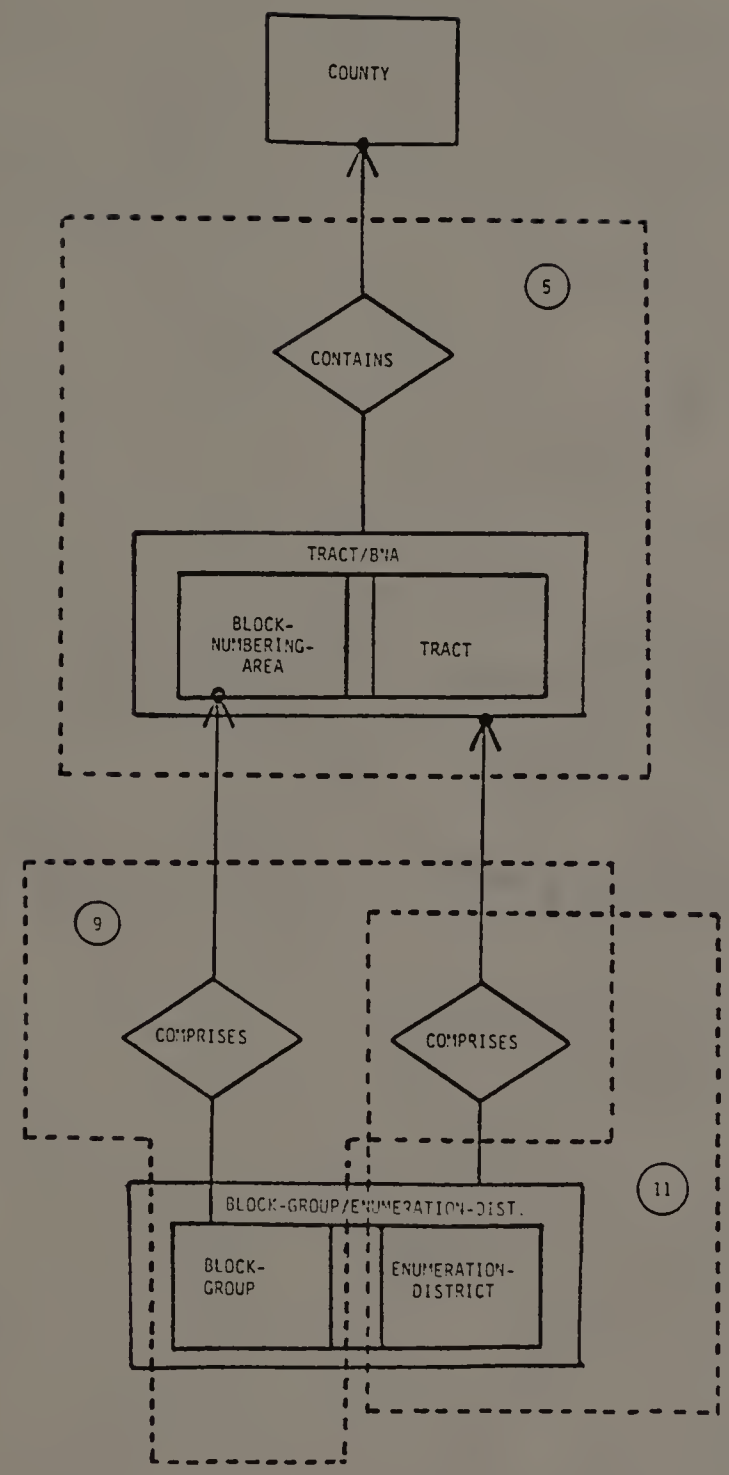


FIGURE 11.4

PL-94 EQUIVALENCE RELATIONS

BLOCK-NUMBERING-AREA, or TRACT/BNA is contained within this single record type.

BLOCK-GROUP and ENUMERATION-DISTRICT exist as separate record types in the PL-94 source structure. In the conceptual schema the respective entity types are joined to form the higher level BLOCK-GROUP/ENUMERATION-DISTRICT entity type, however. Note that the information required to implement the single COMPRISES relationship of TRACT/BNA COMPRISES BLOCK-GROUP/ENUMERATION-DISTRICT demands that two record types be used. The link information not provided by the type 9 records is contained in the type 11 records.

Record types and directories. In studying the equivalence relations we become aware of the GERMS processing inefficiencies that are inherent in the source structure, the PL-94 hierarchy. Recall that a GERMS query requires "downward" processing of the conceptual schema. In other words, our queries ask: "given this (superordinate type) area, which (subordinate type) areas lie within its boundaries?" The queries do not ask: "given these (subordinate type) areas, which (superordinate type) area do they lie within?"

A major problem with the source structure is that it is conducive to "upward" processing only -- each record holds the keys of its ancestors, but it does not hold the keys of its children. Thus given a "subordinate type" area record, we can determine which "superordinate type" area it lies within; but given a "superordinate type" area record, we cannot determine which "subordinate type" areas lie within it. A consequence that holding this configuration has upon query processing is that, for each level of nesting in the query, the entire set of records of the subordinate record type must be searched for each instance of the superordinate record type processed. This processing strategy is difficult to defend.

Because of this situation we suggest that an entirely different physical data structure be implemented. This design, which is the aforementioned target structure, is described here. A major change is that the target structure separates the physical representation of schema entities from that of schema relationships.

Each relationship is represented as a "directory" (see [MART75]) which contains the information required for "downward" processing of the entity types participating in the relationship. Each row of a directory contains a set of pointers which are directed towards the "children" of a

single "parent" in the respective relationship [2]. The rows of a directory are of different lengths since the number of "siblings" in an instance of a given relationship is not fixed. A relationship directory can thus be thought of as being an inverted list.

Figure 11.5 illustrates the relationship directory technique. Here, we see a directory relating to the GERMS relationship COUNTY CONTAINS ELECTION-PRECINCT (figure 11.5b). Each row of the directory depicts an instance of the relationship. Notice that the rows of the table are of different lengths. The relationship instance shown in figure 11.5a happens to relate two election precincts to a county, so the respective directory row contains two elements in addition to the county identifier.

Each entity type of the conceptual schema is, for the most part, represented as a separate record type of the target structure (the exceptions are discussed later). Since the information required to link entities is represented within the relationship directory, a record instance need not hold the key of its ancestry -- it need only hold the identifier and the attributes of the geographic area represented by the record. Thus the records of the target structure are much smaller than are the respective records of the source structure.

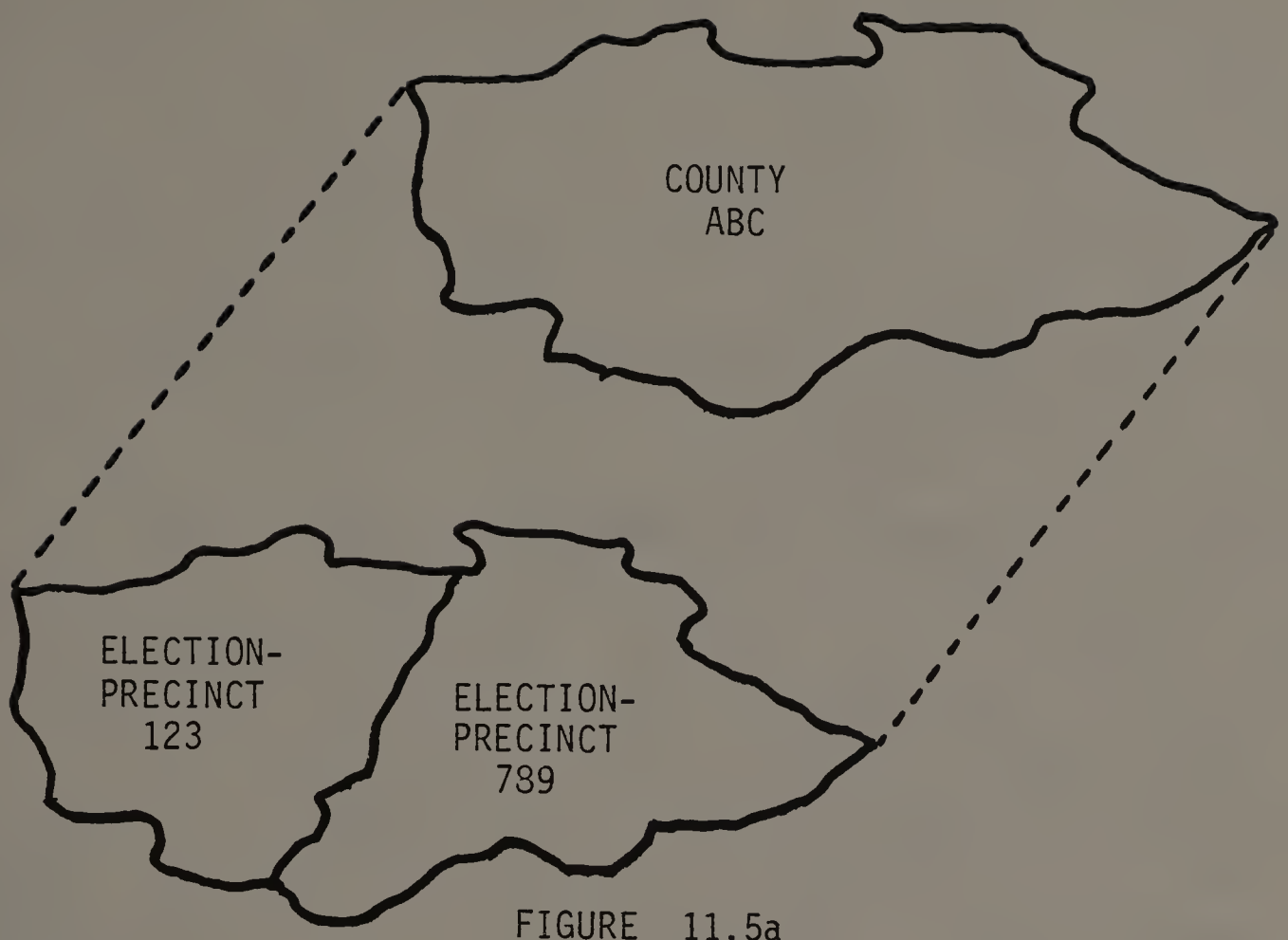
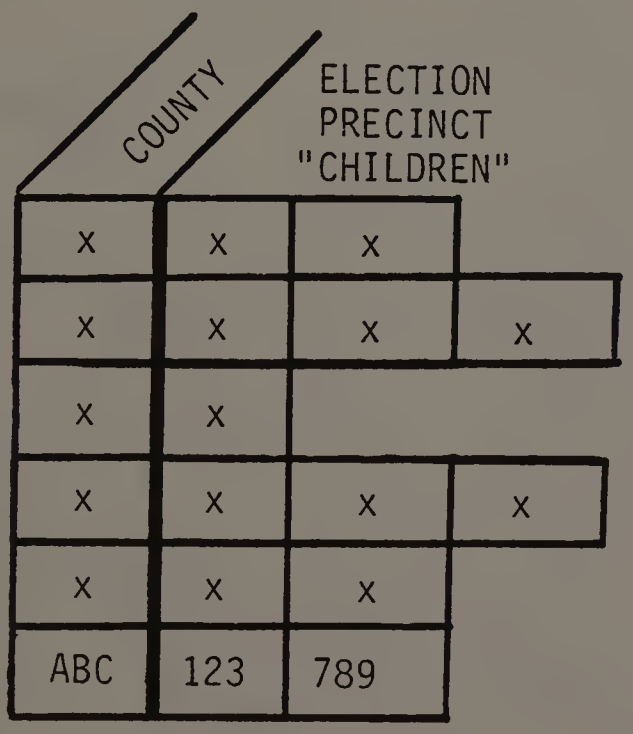


FIGURE 11.5a

A RELATIONSHIP INSTANCE



THE
RELATIONSHIP
DIRECTORY

FIGURE 11.5b

The reduced length of target records does not mean that this revised physical configuration requires less storage than does the hierarchically organized source structure. The data which is "stripped" from the source records, the ancestor key values, is the very data which is used to assemble the relationship directories. Thus, in the process of converting from the source structure to the target structure, we are forming a number of smaller units of information (directories and target records) from larger units (source records).

It should be obvious that the formation of relationship directories is guided by the equivalence relations. These relations indicate which ancestor key value in the source record is required in the formation of a given target relationship directory. In other words, the equivalence relations dictate how the aforementioned large units of information are to be divided and reassembled to form the smaller units.

In light of the way that the conceptual schema is used to define the format of queries, the processing of the target physical structure in response to a query is straightforward. Just as we form chains in the logical structure by moving from entity type to relationship type to entity type, so in the physical structure do we move

from record type to directory to record type [3]. Each object in the conceptual schema has a counterpart in the physical target structure.

Figure 11.6 illustrates how we would process "COUNTY BY ELECTION-PRECINCT" in response to a GERMS-based query. Note that the record instances depicted in this figure are much shorter than are their counterparts in the hierarchical source structure. Once the directories are formed, the ancestry key is no longer required by the record. Note also how, unlike the census file configuration, this revised physical structure is conducive to the required "downward" processing of geographic information.

Geographic "wholeness". In chapter 7 we learned that when the representation of geographic partitionings is "forced" into a hierarchical structure a number of anomalous situations arise. One such situation relates to the case where the boundaries of a child (in the hierarchical structure) area transcend those of an ancestor area. A consequence of this situation is that the child area must be represented in a dissected form; the records of the structure must depict "part-areas."

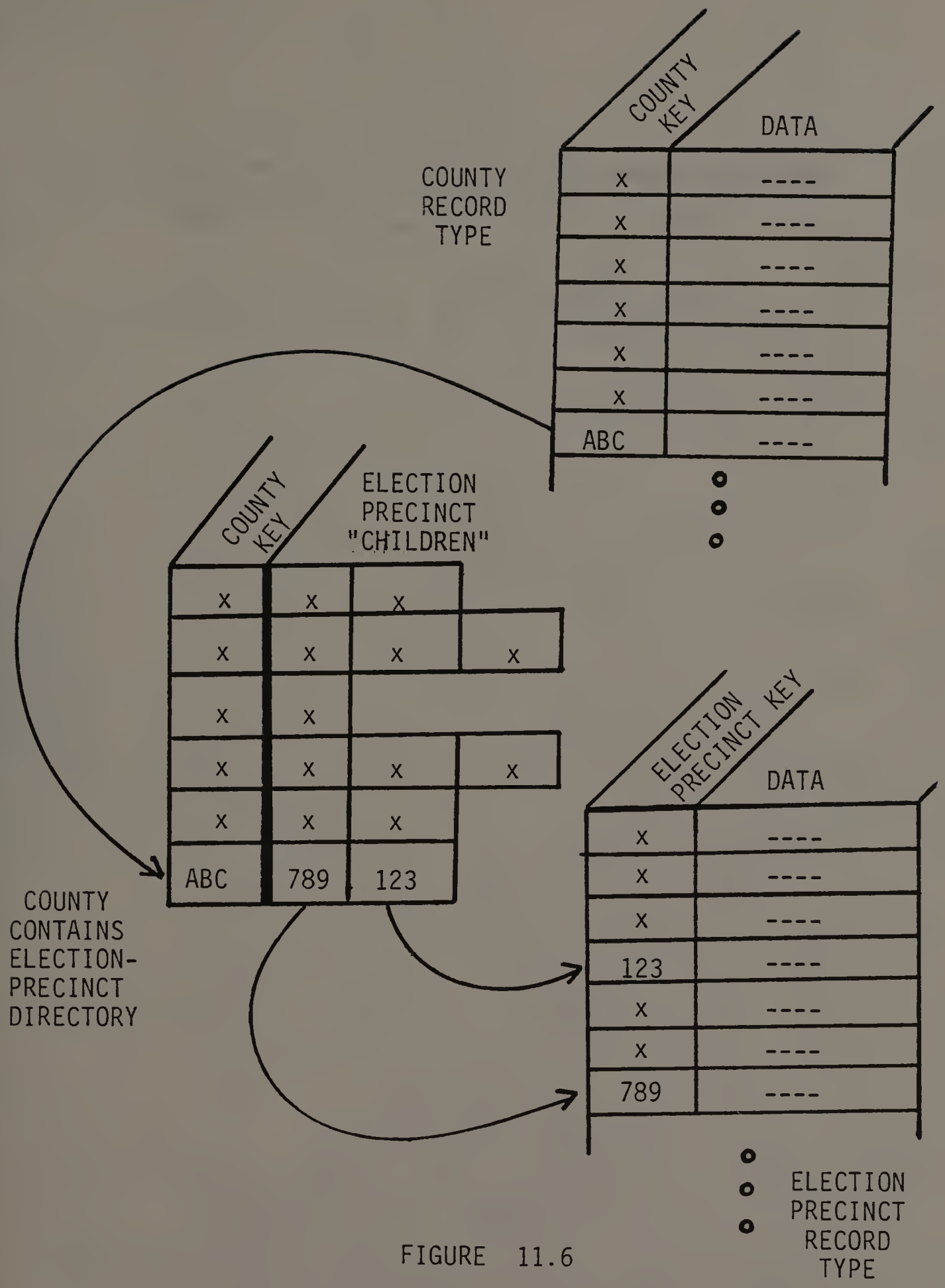


FIGURE 11.6

QUERY PROCESSING

When part-area records are considered in conjunction with their physical hierarchical representation their existence can be rationalized. When the same areas are considered in the light of the true logical or an alternative physical (target) structure, however, their existence is nonsensical. The dissection of a geographic entity arises as a side effect of maintaining a chosen (now obsolete) physical structure. Different hierarchical orderings define different entity partitionings.

We feel that the case for insulating the user from the need to deal with part-areas has been argued sufficiently in previous chapters. Whatever the physical representation of geographic based data, the user should be allowed to work exclusively with logically meaningful objects. Thus, if the physical structure demands that a geographic area be represented in terms of area parts, these parts should be reassembled into a meaningful whole before being presented to the data user. Under the physical configuration suggested here, however, the issue is irrelevant for our GERMS based physical structure never requires that a geographic entity be divided.

Recall that our physical structure is designed to serve the processing needs of the queries that are posed to our geographic information system (GIS). Furthermore,

said queries adhere to the geographic entity-geographic relationship structure of the conceptual schema.

Entity types whose spacial overlap holds a meaningful semantic are joined by a relationship in the conceptual schema. Such entity types, in many cases, hold a "natural" hierarchical association (this is why we found it acceptable to use the terms "parent" and "child" in discussing instances of these pair-wise relationships).

Entity types whose spacial overlap is erratic and meaningless are void of a relationship in the conceptual schema. Such entity types hold an "unnatural" hierarchical association -- one which must be "forced." This fact, considered with the fact that query processing "matches" the structure of our data model, makes it obvious that the need to process a part-area will never arise in the GERMS system. Consequently, there is no need to store such representations.

A germane suggestion, then, is that the data of the part-area records of the source structure be aggregated when assembling the target structure. The part-area records are a manifestation of the now obsolete physical hierarchy, so collapsing these records represents no loss of information from the logical viewpoint. From the physical perspective the savings that result from this

process are twofold; we observe a decrease in both storage space and processing time.

The storage space decrease is obvious. With each aggregation of part into whole, a fixed logical information content is represented by fewer records. We concede that these savings are trivial in light of the massive size of the database.

As for processing time, the betterments are best understood by first considering the alternative scheme -- that of retaining the part-area representation. In this situation each request involving a divided area would demand that more than one record (one for each area part) be accessed. Following these multiple record retrievals, the data would have to be aggregated to represent the original undivided area. This process would be carried out each time such an area is involved in the processing of a query.

Furthermore, there would have to be maintained within the physical structure, some means of identifying and locating those part records that relate to a given divided area. Besides having to know which records relate to area parts (e.g. by using a "part-area flag"), we would also be required to know when the entire set of parts has been retrieved. A pointer chain structure could

be used here (see [MART75]), but this added complexity makes little sense given the needs of query processing.

In summary, then, part-areas are insignificant in the view provided by the conceptual schema. The need for these areas is made obsolete by discarding the hierarchical file structure. As such, the data contained in the respective records of the source structure should be aggregated before being stored in the target structure.

Entity Types and Record Types: Special Cases

In the previous section we indicated that most but not all entity types of the logical structure relate uniquely to record types of the logical structure. In this section we discuss the exceptions. An obvious exception is made in the case of blatant logical redundancy. By blatant logical redundancy we mean that two objects represented within the logical structure depict the exact same physical object. An example is provided by the IS-A relationship [4].

Representing IS-A. The IS-A relationship is rare; the majority of geographic relationships are of the CONTAINS or COMPRISES variety. The PL-94 schema of figure 11.2 serves as testimony to this fact. Still, when the need arises, the IS-A relationship illustrates an important semantic of data abstraction. From the logical perspective the IS-A relationship connects two distinct geographic entity types; one representing a more general case of the other. Again from the logical perspective each of these two entity types represents a number of entity instances. Thus we can conceptualize special case entity instances and general case entity instances.

When we consider the incarnation of these entity instances, however, we realize that the special case and the general case instances depict the same physical objects. It is only the way of "viewing" the objects that changes. In these different views the nature of the geographic relationships may change, but the actual attributes of the respective areas do not change. Thus the two views can be physically represented by a single set of data records. Neither the value nor the semantic of any individual datum changes.

The PL-94 conceptual schema of figure 11.2 does not provide us with an example of an IS-A relationship so an example will be drawn from the STF1B file which was introduced in chapter 7. Figure 11.7 shows a small portion of the STF1B conceptual schema. From this schema we see that the tracts which comprise SMSAs (called urban tracts*) are completely covered by block groups. The entity type URBAN-TRACT* is a special case of the more general type TRACT; the distinction arising from the fact that general tracts are not necessarily covered by block groups (the type TRACT holds some relationships which are not shown here).

Now, since any entity type of the conceptual schema can be referenced within a query, we may request data in terms of urban tracts* or in terms of tracts. Furthermore, because the queries must abide by the structure of the data model, we see that a query format of the type "URBAN-TRACT* BY BLOCK-GROUP" is a legitimate request, while the format "TRACT BY BLOCK-GROUP" is not.

When we consider real world counterparts of instances of these entity types we realize that each geographic area which is an urban tract* is also a tract. The attributes of the area are the same in both lights. The only difference is that, when viewed from the urban

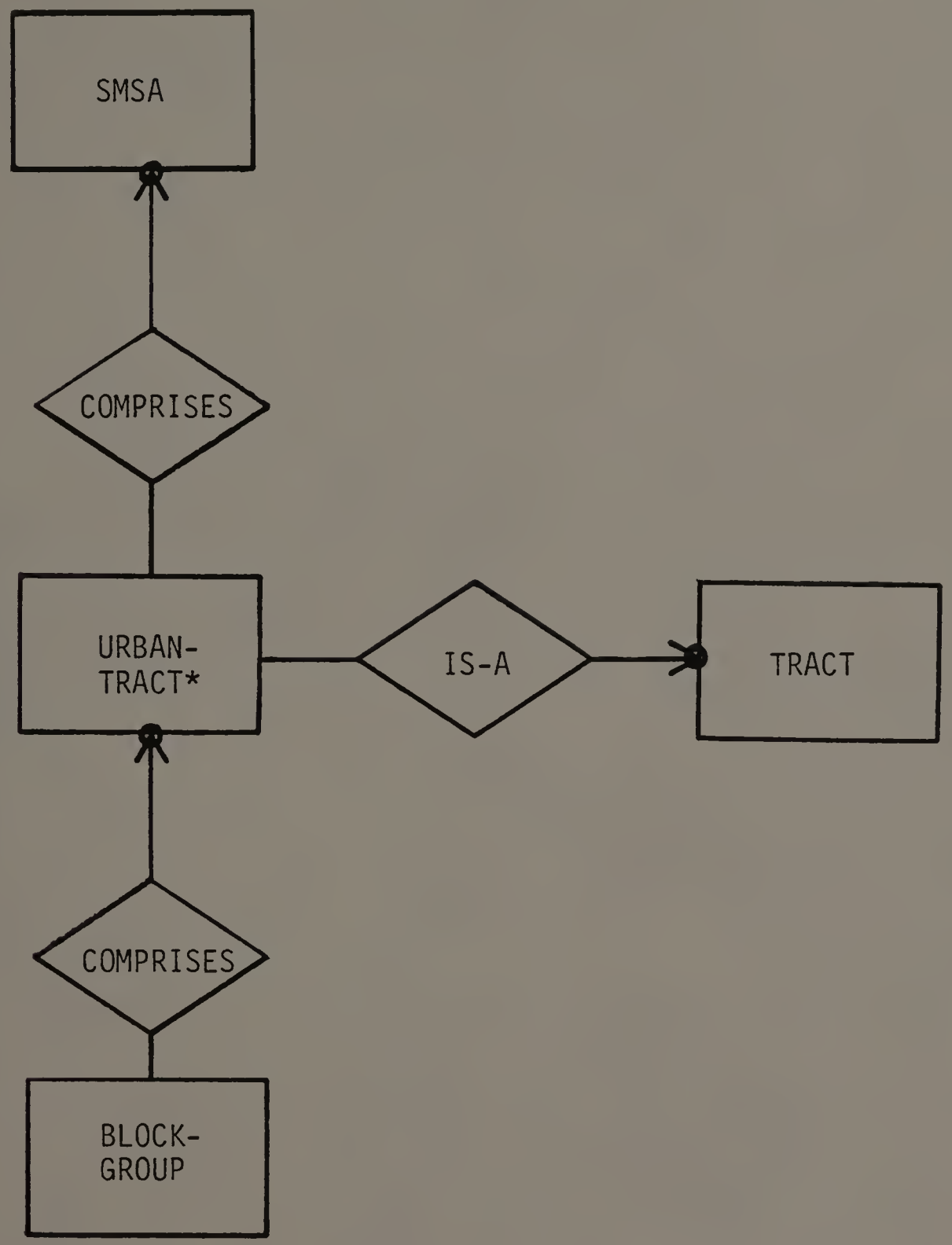


FIGURE 11.7

A PORTION OF THE STF1B

CONCEPTUAL SCHEMA

tract* perspective, the area is related to an SMSA and to a set of block groups. In terms of physical storage, then, there is no need to store the geographic based data twice; a single data record can represent the geographic area from both perspectives.

This concept is illustrated in figure 11.8. Here, one user (Able) is considering urban tract* ABC, while a second user (Baker) is considering tract ABC. These two area names relate to the same record description which is the means through which the DBMS locates the physical data. This single record description is in turn related to a single physical data record.

Since the URBAN-TRACT* and the TRACT logical objects share a common physical representation, only one record type, say type TRACT, is required to serve both logical entity types. A consequence of this physical representation is that a relationship directory is not required for the IS-A relationship type. The IS-A relationship is never used to traverse entity types in an information retrieval request; it is used to create new entity types which can be referenced in a retrieval request.

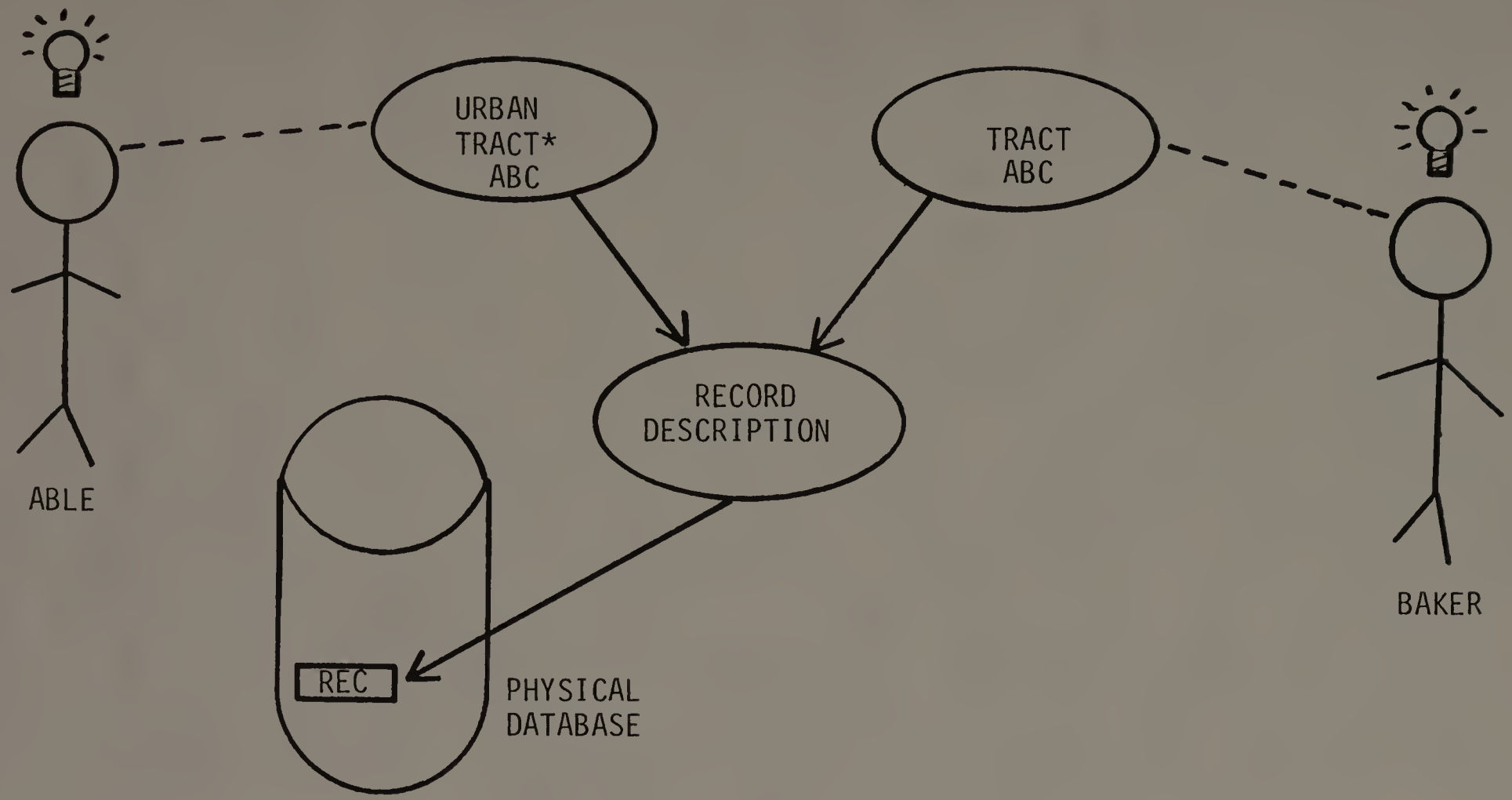


FIGURE 11.8

A SINGLE RECORD CAN BE

VIEWED FROM TWO PERSPECTIVES

Figure 11.9 illustrates the physical configuration that relates to the conceptual schema of figure 11.7. Note that the SMSA COMPRISES URBAN-TRACT* relationship directory references only those tracts which lie within an SMSA. Thus only a portion of the type TRACT records are pointed to from this directory. Similarly, the remaining directory only points to those block groups which lie in an SMSA. A directory which relates to general tracts (vis. urban tracts*) would point to all records of type tract, including those of the "urban variety."

From studying figure 11.9 it becomes obvious how a query of the form "SMSA BY URBAN-TRACT* BY BLOCK-GROUP..." would be processed. A problem arises, however, when we consider a query of the form "URBAN-TRACT* BY BLOCK-GROUP..." Here, we must "enter the chain" of nesting at the urban tract* level. The problem is that, unless SMSA records are considered first, there is no way of identifying which tract records are also urban tract* records.

One solution arises when we consider that the SMSA COMPRISES URBAN-TRACT* directory contains the identifiers of these special records within its inverted list structure. These identifiers are not stored in sequential order, however. Consequently, we suggest that a separate

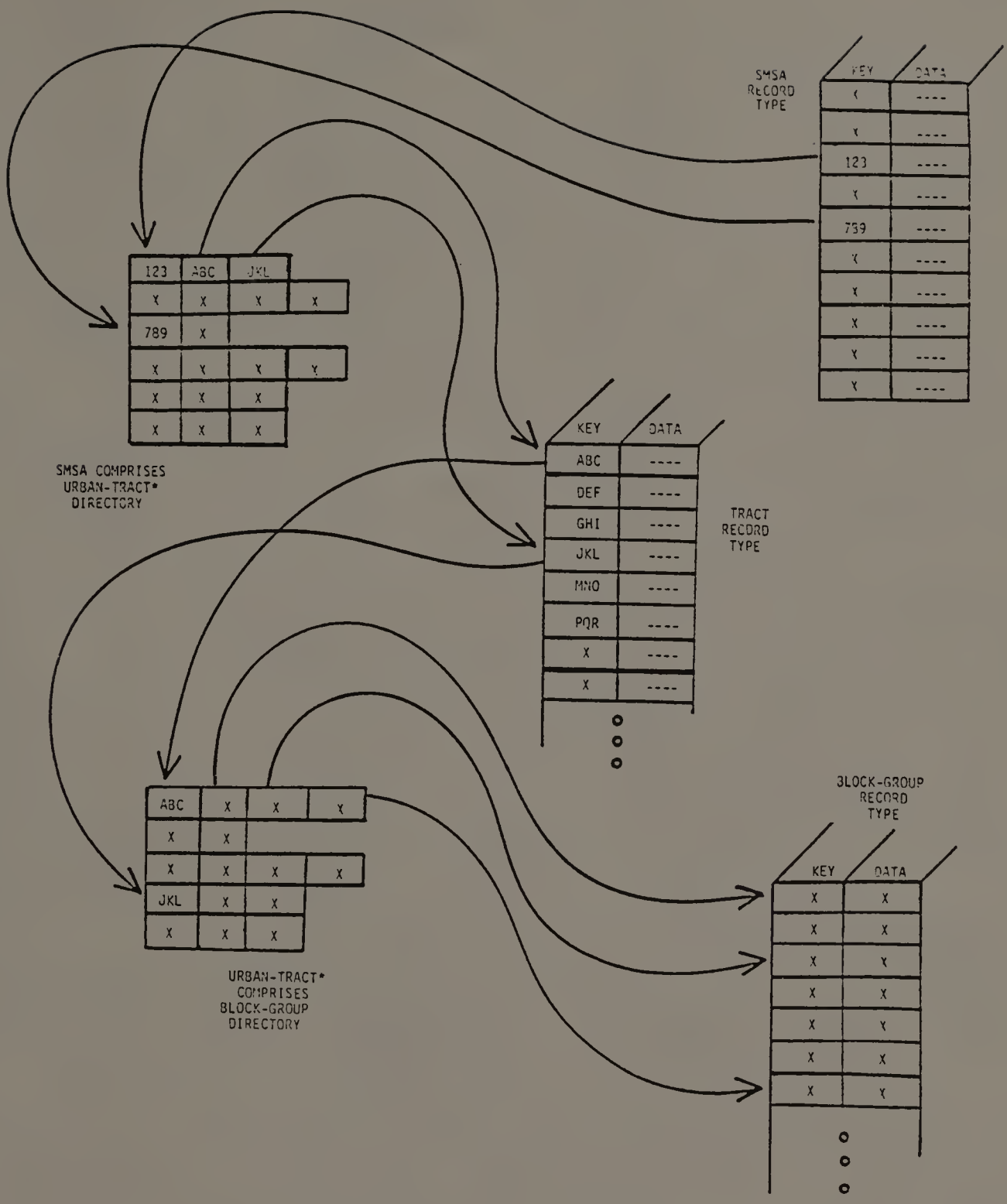


FIGURE 11.9

THE PHYSICAL CONFIGURATION OF
THE FIGURE 11.7 SCHEMA

index be added for each "special type" of entity. This technique is illustrated in figure 11.10.

NOTE: Up to this point we have ignored the issues surrounding the access of the individual records of a given record type. The rationale behind this ignorance of detail is that the resulting discussion was simplified. The mere reference to pointers, as made by our diagrams, implies the assumption that, given the key of a record, we can retrieve the record. Indeed, each standard record type may have a dedicated index through which its records can be accessed. For example, each record type of our physical structure may represent a separate ISAM file. The index described in the preceeding paragraph is an additional index, however. It is created to supplement whatever standard record locating technique is implemented.

Representing EITHER-OR. A physical representation issue which is quite similar to that of the IS-A relationship has to do with the treatment of "EITHER-OR-joined" entity types. An example of this is found within the PL-94 conceptual schema (figure 11.2). The relevant portion of this schema is repeated in figure 11.11. Here we see two EITHER-OR associations whose "higher level" entity types

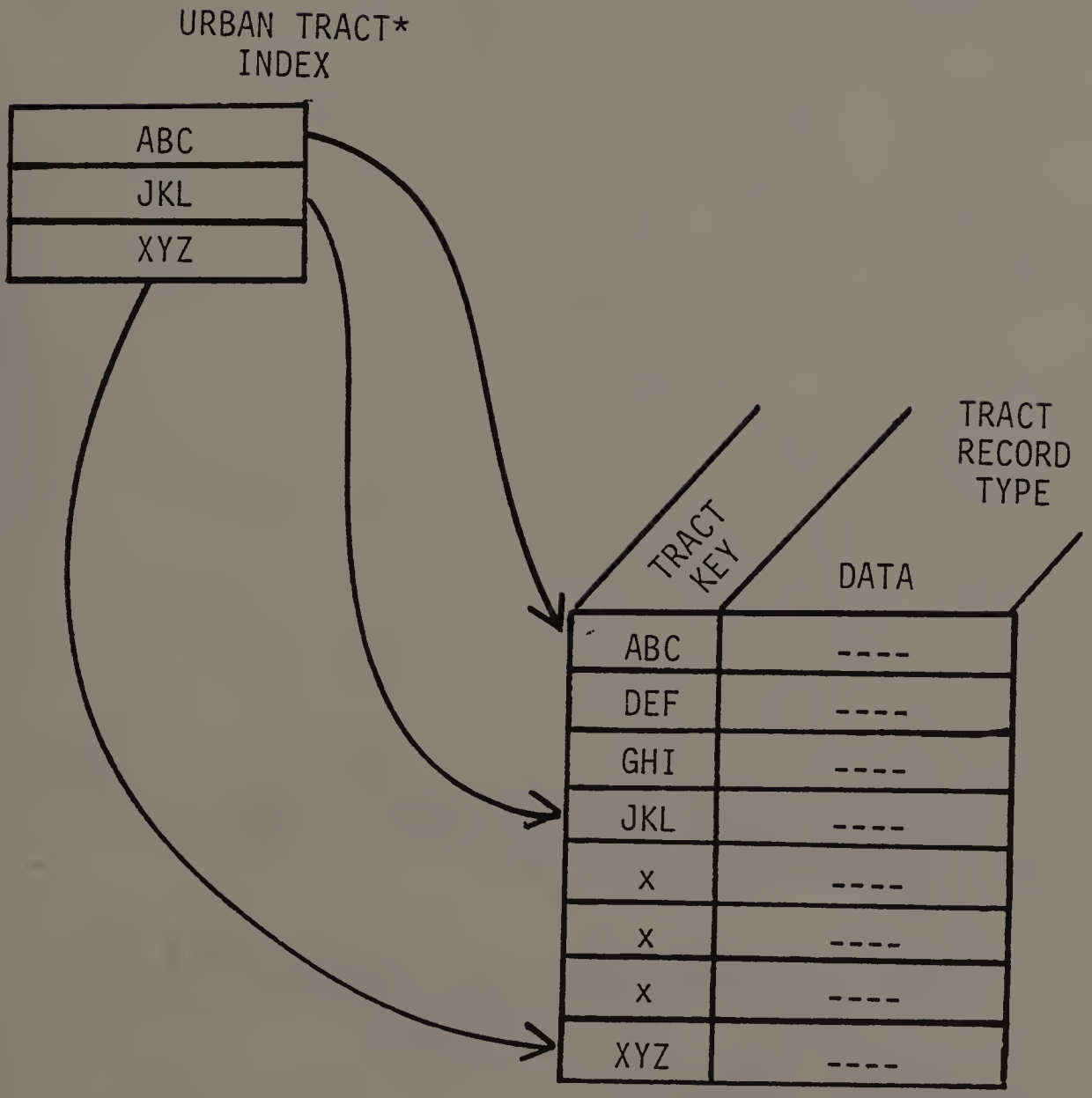


FIGURE 11.10

A "SPECIAL CASE" RECORD INDEX

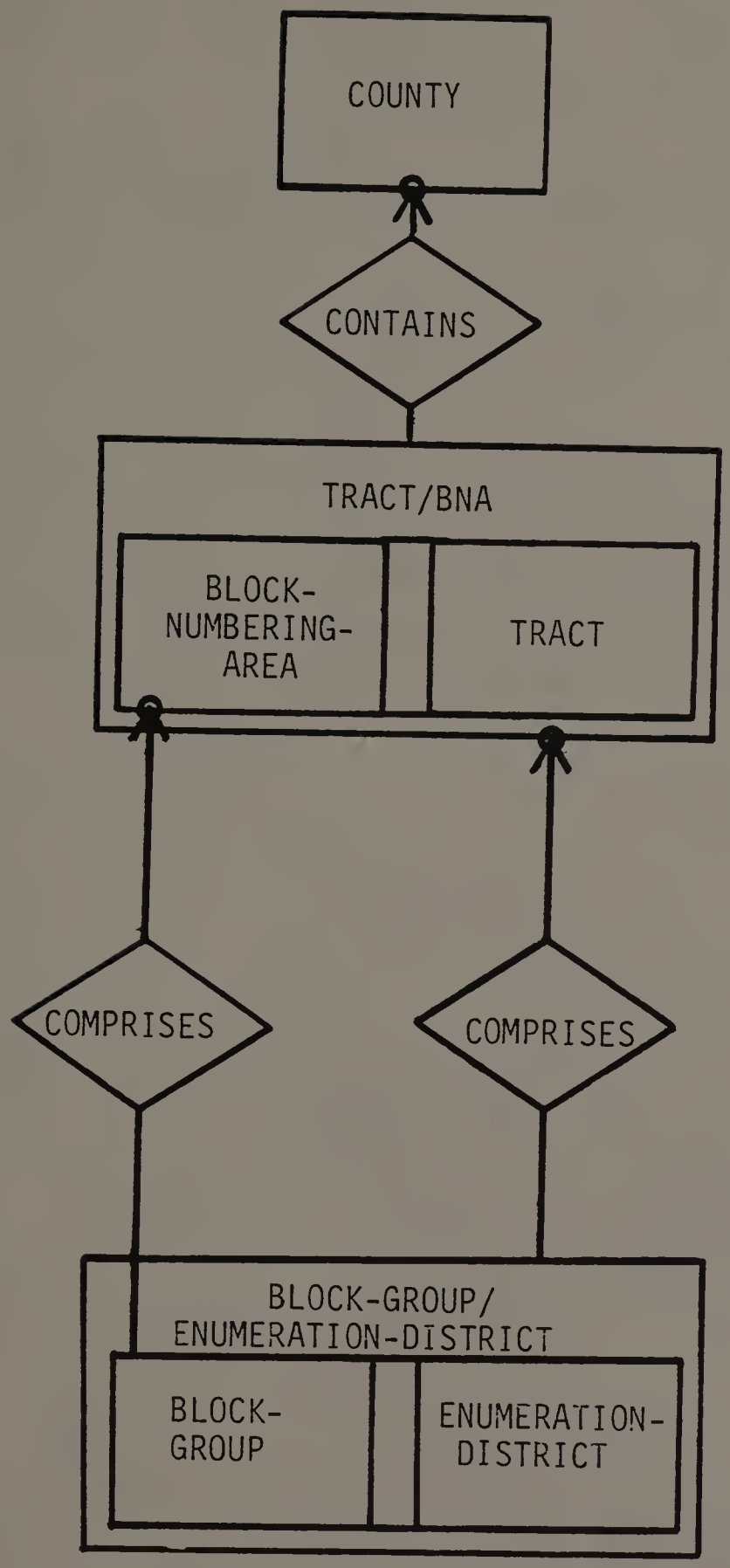


FIGURE 11.11

A PORTION OF THE PL-94 SCHEMA

are connected by a COMPRISES relationship. Also, two of the "lower level" entity types are connected by a separate COMPRISES relationship.

As is the case with the IS-A relationship, instances of different entity types refer to the same real world geographic areas when the EITHER-OR association is used. In this situation each instance of a "lower level" entity type (e.g. TRACT or BLOCK-NUMBERING-AREA) is also an instance of the related "higher level" type (i.e. TRACT/BNA). Thus, for the same reason as was presented previously, a single record type can serve all of these entity types.

Since the "lower level" entity types represent "special cases" of their "higher level" type, the record type should relate to the more general higher level entity. Thus, in relation to our example schema, we would observe a record type TRACT/BNA, and a record type BLOCK-GROUP/ENUMERATION-DISTRICT in the physical database. Obviously, each record instance would relate to only one simple geographic area type (i.e. tract, block numbering area, block group, enumeration district).

From the figure 11.11 schema we see that a query of the form "COUNTY BY TRACT/BNA..." is a legitimate request. The elements of the physical structure which

would be used to process this request are obvious. Specifically, they are:

1. A COUNTY record type which contains county instance records

2. A TRACT/BNA record type which contains both tract instance records and block numbering area instance records

3. A COUNTY CONTAINS TRACT/BNA relationship directory which connects county records to their respective tract and block numbering area records (note that every instance of both record types is connected here)

In responding to this query the system should indicate to which (lower level) entity type each record relates. In other words, the user should be told, for each record retrieved, whether the data applies to a tract or to a block numbering area. This is easily accomplished by storing a "type flag" within the contents of the each record. When a record is retrieved, this flag is used by the system to determine the lower level type to which the

record applies. This is then included in the information presented to the user. The use of this type flag is illustrated in figure 11.12 (Note that since this flag is used to represent a binary attribute which is for system use, the flag can be represented with a single bit).

Figure 11.11 (above) indicates that we may pose a query of the form "TRACT/BNA BY BLOCK-GROUP/ ENUMERATION-DISTRICT..." to our system. Again the relevant objects of the physical structure are obvious:

1. A TRACT/BNA record type

2. A BLOCK-GROUP/ENUMERATION-DISTRICT record type

3. A TRACT/BNA COMPRISES BLOCK-GROUP/ENUMERATION-DISTRICT relationship directory

Note that this relationship directory references <u>all</u> instances of these record types.

From figure 11.11 we also observe that we may request "BLOCK-NUMBERING-AREA BY BLOCK-GROUP..." Now, we know that the same record types would be used to process this query as would be used to process the previous query. An interesting question is: Can we use the same relationship directory? The answer is yes; the pointers

TRACT/ BNA KEY	DATA	TYPE FLAG
ABC	----	TRACT
DEF	----	TRACT
GHI	----	BNA
JKL	----	TRACT
X	----	BNA
X	----	BNA
X	----	BNA
X	----	TRACT

FIGURE 11.12

RECORD TYPE TRACT/BNA

required to process TRACT/BNA BY BLOCK-GROUP/ENUMERATION-DISTRICT are a superset of the pointers required to process BLOCK-NUMBERING-AREA BY BLOCK-GROUP.

With regards to this latter query, since we "enter the chain" by selecting BLOCK-NUMBERING-AREA records only, the pointers will direct us to those rows of the relationship directory which deal with block numbering areas. Block numbering areas, by their very nature, are covered by block groups, so these rows of the directory will point exclusively to BLOCK-GROUP records. Thus, once the chain is entered, the processing can continue without problems.

The issue of how this chain is entered (i.e. how the BLOCK-NUMBERING-AREA records are selected) remains unclear at this point. Given that each record of type TRACT/BNA contains a type flag, the system could inspect each record and process those which, as indicated by the flag, relate to a block numbering area. A more time efficient way of handling this problem is to apply the same technique as was suggested for use with the special case records of the IS-A relationship. Thus, we suggest that a dedicated index be included for each "lower level" entity type of the EITHER-OR association.

The TRACT/BNA record type would therefore require two special purpose indexes. One such index would direct the search to the records which relate to tracts, while the other would direct the search to the records which relate to block numbering areas. The targets of these indexes do not overlap (see figure 11.13).

Representing attribute areas. Recall from chapter 7 that census files often contain data about geographic areas to which no specific data records are dedicated. We call such geographic areas "attribute areas" because the applicability of a record to the area type is indicated by an attribute in the record. Rather than being stored in a single record, the data relating to an attribute area is scattered throughout the file. Meaningful information about an instance of an attribute area must therefore be derived by collecting and aggregating the data from a number of scattered records.

Each attribute area is represented as an ordinary geographic entity type in the GERMS conceptual schema since the unusual physical representation of the data does not affect the logical data structure. Thus, as is always the case, the complexities of physical storage are not evident from the conceptual schema. Since all information

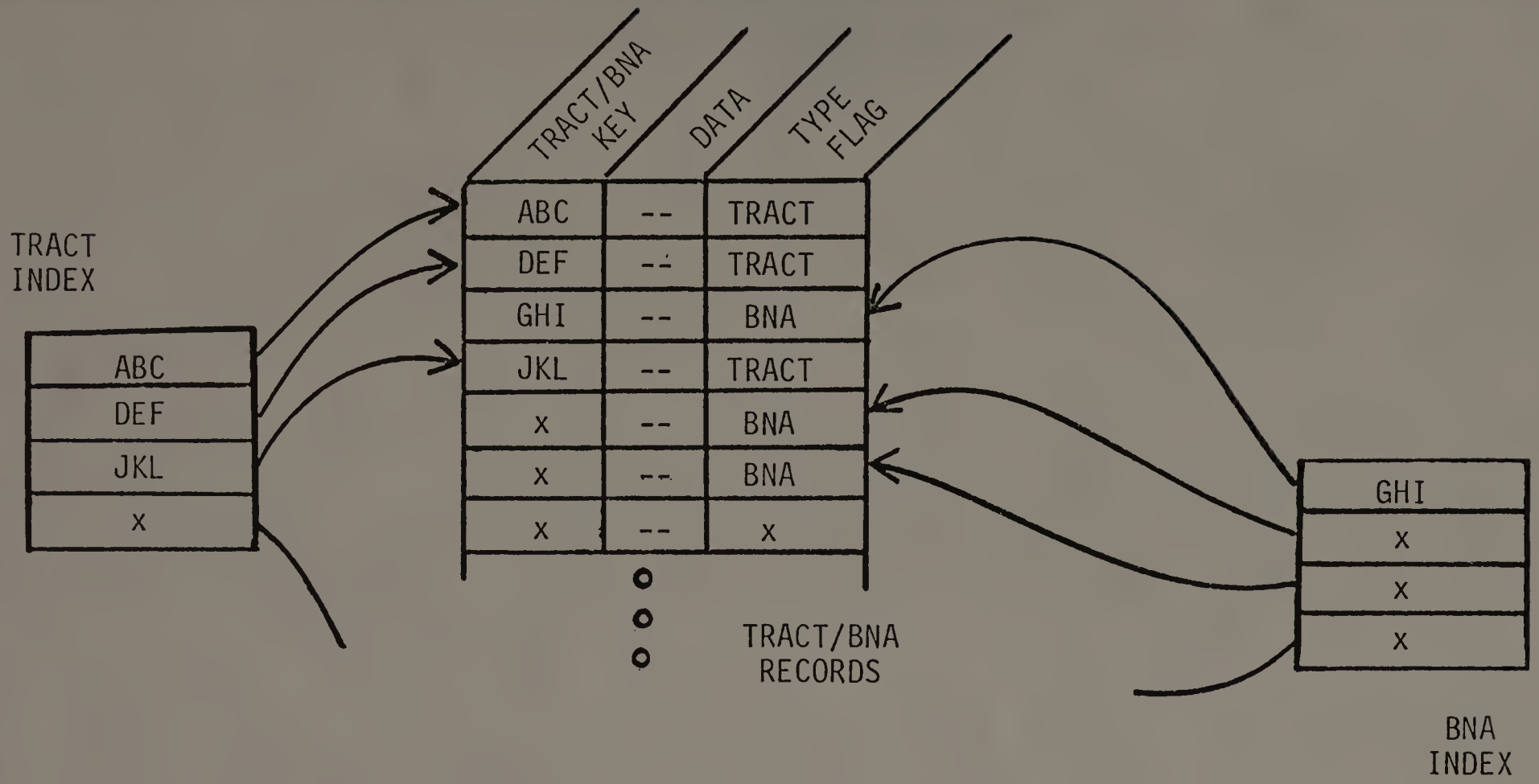


FIGURE 11.13

USING INDEXES WITH THE EITHER-OR ASSOCIATION

provided by our system is presented to the data user from the perspective of the data model, the DBMS has the responsibility of insuring that the system output represents a complete "reassembled" geographic object when an attribute area is referenced in a query.

The issue at hand in this subsection has to do with the physical representation of attribute areas. There are two basic approaches to this:

1. The attribute area instances can be assembled prior to storage. This involves collecting and aggregating the data records which relate to attribute areas and forming new records from this aggregated data. The data records which relate to a given type of attribute area would be combined to form a new record type to be stored in the usual way.

2. The attribute area instances can be reassembled during query processing. This demands that there be some means of collecting the scattered parts of an area instance (sequential scan is not acceptable for obvious reasons).

If the frequency of access of attribute area data is high, the former alternative is the obvious choice. True, this configuration requires greater storage space (more data stored with constant information content), but the processing time is much lower in comparison to the latter. If for some reason the second alternative is selected, we are prepared to suggest two physical representation schemes.

For the purpose of illustration we will rely on the chapter 7 example of urbanized areas (from census file STF1B). Here, the file contains a record type BLOCK, the instances of which obviously relate to block areas. Some of these BLOCK records hold the key value of an urbanized area of which the block is a part. Thus the entity type URBANIZED-AREA is an attribute area. The problem at hand is: Given a retrieval request for an urbanized area, how can the appropriate BLOCK records be collected?

The reader may be thinking that we can make use of physical position by storing contiguously the BLOCK records which relate to an urbanized area instance. This juxtaposition technique is unacceptable since the records of type BLOCK must be <u>stored according to block key;</u> not urbanized area key.

The first acceptable scheme is based on pointer chains. Here the records which relate to a given instance of an attribute area are combined to form a chain. Each non-terminal element is pointed to by the previous element and points to the next element. Thus, once the first chain element (head) is accessed, all other elements can be located by following the pointers. An index can be used to identify the head of the chain. This is illustrated in figure 11.14.

The second scheme uses an inverted list structure which is similar to that of our relationship directories. Here, the directory holds the information required to locate the scattered parts of an attribute area. Each row of the directory relates to an area instance. The elements of a row point to those records which hold data relating to the instance. This technique is shown in figure 11.15. Note that the only difference between the two schemes is that in the former, the pointers are stored within the records, while in the latter the same pointers are stored within the index (directory).

As a final word we would like to clarify that each of these two schemes must be used in conjunction with a special processing procedure. This procedure has the responsibility of collecting and aggregating the data

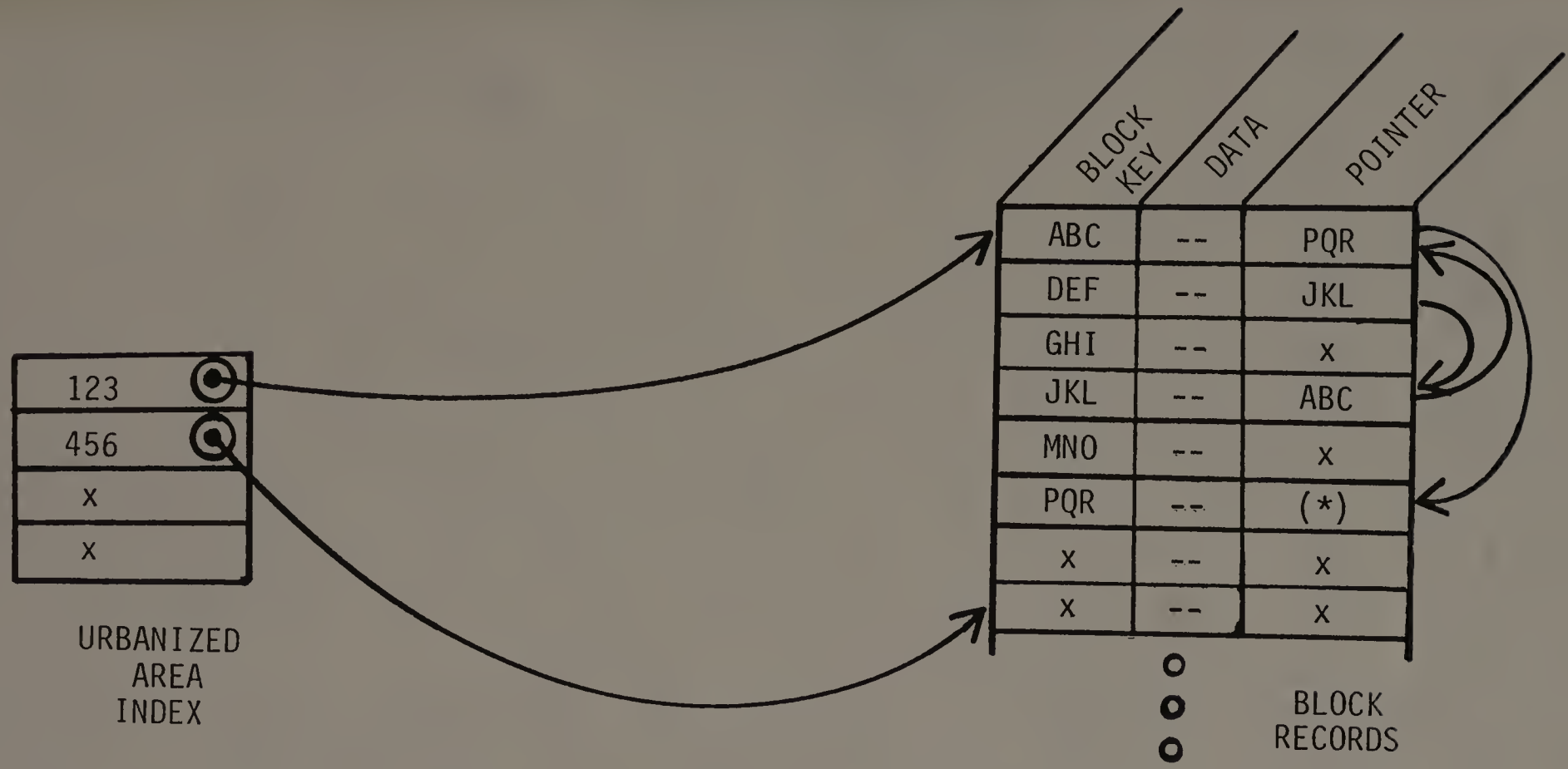


FIGURE 11.14

STORING ATTRIBUTE AREAS

AS CHAINS

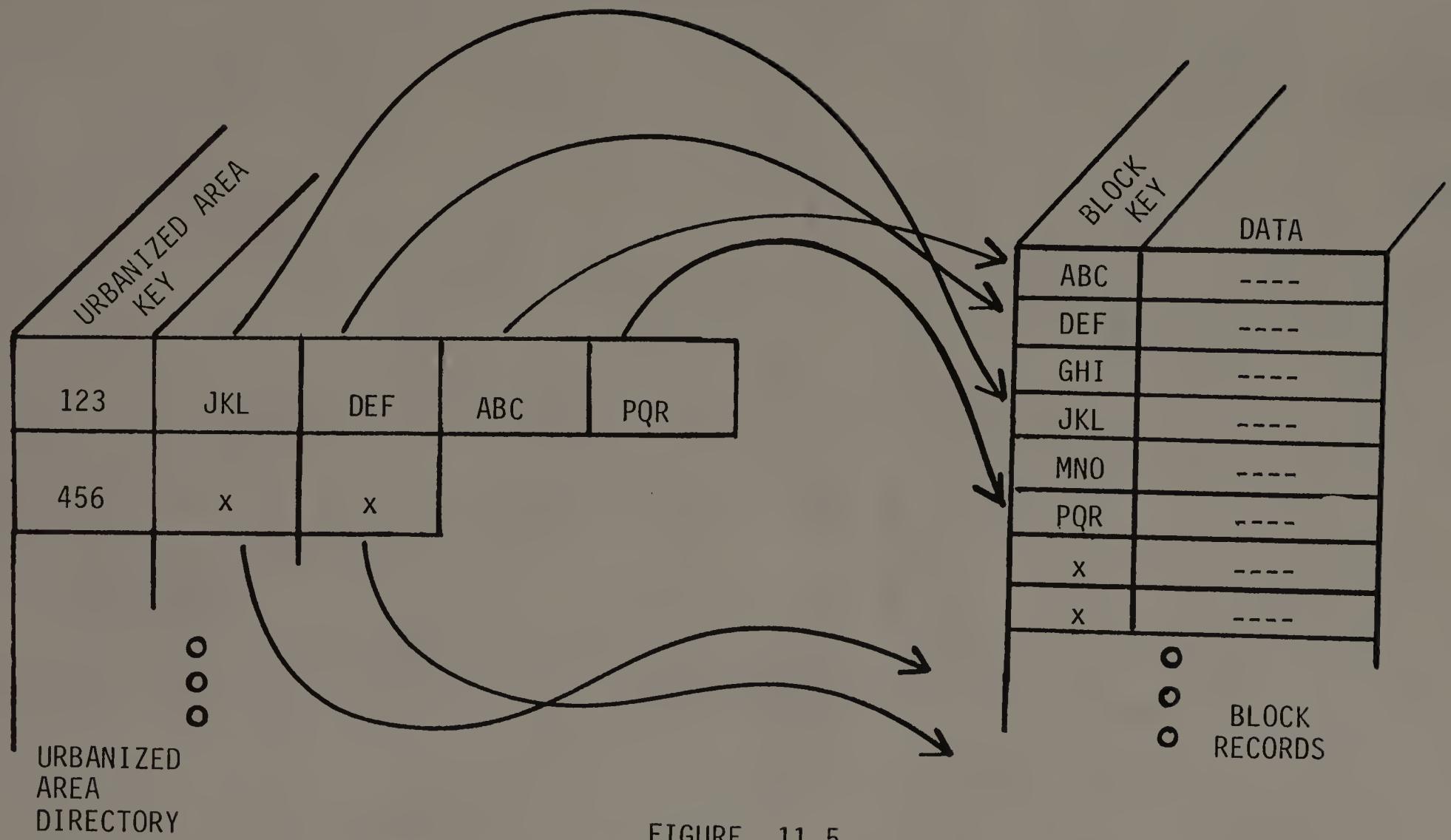


FIGURE 11.5

STORING ATTRIBUTE AREAS AS AN INVERTED LIST

records before the information is presented to the user. The structure of such a procedure is straightforward and is not discussed here. The execution of the procedure is "triggered" automatically upon reference to a logical entity which is physically stored as an attribute area. In this way the user is "insulated" from the complexities of the physical storage scheme; the final system output "appears" the same as that of any ordinary (non-attribute area) geographic area.

Summary

Before the database of our geographic information system can be constructed we must equate the data of the source structure to the objects of the schema structure. The equivalence relations are used for this purpose. The objects of the target structure correspond directly to the objects of the schema, so the equivalence relations serve to detail the formation of the physical database objects.

Unlike the source structure, the target structure separates relationship data from entity data. Consequently, the equivalence relations indicate that the relationship data can be "stripped" from the source

records upon populating the database. This very data is used to form the relationship directories. The GERMS deals exclusively with "natural" geographic entities, so geographic areas which are represented in "dissected" form within the source structure can be reassembled when creating the target structure.

The conceptual schema employs logical redundancy, but there is no need for redundant representation of geographic areas in the physical database. This ability is achieved through the use of multiple indexes which are directed towards a common record type. The user is insulated from this complexity, however. (S)he need only reference the geographic objects which relate to her/his "view" of geography; the system has the ability to locate the required physical data.

In the next chapter we consider the makeup of the logical database -- that part of the GIS which relates to its "knowledge" of the conceptual schema. The next chapter also describes how the logical database is linked to the physical database. This link is required since query processing involves an integration of the physical with the logical domain. Queries are posed in terms of the latter, but responses are formed by processing the former.

FOOTNOTES

[1] The terms "source" and "target" are so named with regards to the process of loading the database. From the perspective of this process, census data is the input. The structure of the census file is the "source" of the input. The purpose of this process is to restructure the data to match the framework of the database. Thus the revised structure is the "target" of the process.

[2] The jargon of tree structures is used here for the purpose of clarification only. We do not mean to imply that a strict hierarchy is being instituted as the target structure.

[3] This description is an obvious oversimplification of query processing. Our output "report" must have a well-defined nested structure: that of a "preorder traversed" tree (see [PAGE78]). Consequently, the required elements of all directories used in a query must be assembled into an appropriately ordered queue before any data records are accessed. Then, by retrieving the records in queue order, the elements of the output report are properly nested.

[4] The entire GERMS model can be considered to be logically redundant since areas represented by entity types overlap. In this case, however, we are concerned with the situation where two instances of different entity types can be equated.

C H A P T E R X I I

THE LOGICAL DATABASE

The techniques and capabilities of our GIS (geographic information system), as discussed to this point, presuppose that the system holds a "knowledge" of the conceptual schema. When the objects and connections of the schema are referenced in user --> system queries, the system "understands" the references; when the schema structure is contradicted in a query, the system detects the error. Also, the response to queries, or system output, is presented "in terms of" the structure of the conceptual schema. The schema is thus the system's interface with the user. It represents the "common ground" shared by the human user and the database machine.

In this chapter we continue to discuss the makeup of our GIS. Specifically, we detail the structure of the logical database. We will see that this entity comprises a number of component parts, all of which are "invisible" to the data user. Some of the component parts serve to provide a linkage to the physical database. This linkage is required if data is to be retrieved properly. Neither the logical database nor the physical database can operate independently; they must cooperate in meeting the demands

of the data user.

Conceptual Schema: Internal Representation

A conceptual schema, or more generally a data model, is not a tangible object. It is not a GERMS graph or a set of GERMS phrases; these, rather, are representations of the schema. The schema is thus the information that is contained in these representations. Since the information content of the graphical and the lingual form are the same, we say that they are "equivalent" representations. Given the schema graph, we can produce uniquely the set of GERMS phrases. Given the set of phrases, we can construct the schema graph.

In this section we discuss how the GIS can hold a representation of the conceptual schema. Also, we consider how the system can "interpret" this schema representation. The GERMS schema graph avails itself to the human users of our GIS because the graphical representation is easily understood by this community. To the data user the GERMS graph is a concise presentation of the information from which the schema is formed. The GERMS lingual form is also well suited for employment by humans.

It is obvious that our GIS cannot make sense of the schema graph as can the user community. Still, our techniques demand that the system has access to the schema. What we seek, then, is some form of representing the structure of the schema which can be made use of by the system. We do not desire a different schema, only a different representational form -- a form which is equivalent to the schema graph.

We call such a representation an "internal form representation" because it is contained in, or internal to, the information system. When viewed from outside of the information system, the component is neither observable nor important. Thus the user need not consider the internal form; (s)he need only be confident that the system "understands" her or his references to the objects of the schema graph.

In considering the nature of the information which must be "captured" by a schema representation we will refer to the schema graph. First of all, recall that the schema graph is made up of two distinct kinds of objects: entity types and relationship types. The fact that one kind of object is represented by a rectangle, and the other by a diamond is of no concern to us here. We must only consider that, as presented by the graph, the two object kinds are distinguishable.

Second of all, the entity type names depicted in the schema graph are data set dependent. Thus the entity type names are attributes of the specific schema (not of the modeling formalism) and, as such, the names must be captured in our internal representation.

The relationship types, on the other hand, are data set independent. The four relationship types; CONTAINS, COMPRISES, IS-A, EITHER-OR; are inherent in the underlying modeling formalism (note that we will consider the EITHER-OR association type to be a relationship type for the time being). Because of this, our internal form representation does not rely on a knowledge of the relationship type names. It must only distinguish that there are four different relationship types.

The relationships in a schema are always pair-wise, i.e. they link exactly two entity types. Also, a pair of entity types is joined by at most one relationship. Furthermore, the relationships are directed in that the ordering of entity types in a relationship is significant.

With these facts in mind we realize what information the internal form schema must hold. First, given an ordered pair of entity types, it must be able to indicate whether a relationship holds between the types. If a relationship does hold, it must further indicate (1) which

of the four types is the relationship, and (2) what is the direction of the relationship. This is the extent of the information provided by the schema graph.

A feature of the GERMS formalism which adds a complexity to the internal form representation stems from the fact that the presence of some relationships in the structure imply the presence of other relationships elsewhere in the structure. For example, if A COMPRISES B, and B CONTAINS C, then it is implied that A CONTAINS C.

The schema graph need not represent explicitly the CONTAINS relationship between entity type A and entity type C because the implicit presence of the relationship is derived "in the user's mind." The derivation is a function of the well-defined semantics of the relationships. In other words, given the "meaning" and the structure of the explicit relationships, the fact that A CONTAINS C is obvious to the user. As such, this relationship could be referenced in a query.

If the internal form representation is to be an equivalent representation, then, the GIS must somehow "know" that this implicit relationship exists. In general, given the relationships that are explicitly depicted in the schema graph, the system must have the capability of deriving those relationships which are implicit in the structure.

In the subsections that follow we suggest how an internal knowledge of the conceptual schema can be created, stored, and used by our geographic information system. These techniques can form the basis of a "GERMS front end" to an existing DBMS (database management system). It is assumed that the DBMS supports (1) multiple record types, (2) record indexes, and (3) inverted list structures (directories). The intricacies of these DBMS features are not considered in detail.

The schema matrix and symbol table. If a GERMS schema makes use of N entity types, then the entire structure of the schema can be represented by a single N x N matrix. Consider a square matrix named SCHEMA where the term SCHEMA(i,j) signifies that element residing in the ith row and the jth column of the matrix. Now assume that the N entity type names of the schema graph are assigned distinct sequence numbers: the integers 1 through N. Since a sequence number can represent a specific entity type, an element of the N x N matrix SCHEMA can denote a specific pair of entity types. Thus the element SCHEMA(i,j) can denote the pair "ith entity type, jth entity type."

Every GERMS schema distinguishes at most four relationship types: CONTAINS, COMPRISES, IS-A, and EITHER-OR. These can be symbolized by the integers 1, 2, 3, 4, respectively.

Now, given any ordered pair of entity types, a GERMS schema representation must indicate whether a relationship holds between the types. In the case where a relationship does hold, the schema representation must describe the relationship in both type and direction. Our matrix SCHEMA can indicate this information easily. Given a pair of entity types, say the ith type and the jth type, a non-blank entry in the ith row and the jth column of the matrix (SCHEMA(i,j)) indicates that a relationship holds between the relationship pair. The relationship type and direction are indicated by the value of this non-blank entry.

If the ith entity type is named I, the jth entity type is named J, and the kth entity type is named K, then the internal form schema representation is as follows:

1. If I CONTAINS J then

    1. SCHEMA(i,j) = 1
    2. SCHEMA(j,i) = 0

2. If I COMPRISES J then

    1. SCHEMA(i,j) = 2
    2. SCHEMA(j,i) = 0

3. If I IS-A J then (note the order here)

    1. SCHEMA(i,j) = 0
    2. SCHEMA(j,i) = 3

4. If I IS EITHER J OR K then

    1. SCHEMA(i,j) = SCHEMA(i,k) = 4
    2. SCHEMA(j,i) = SCHEMA(k,i) = 0
    3. SCHEMA(j,k) = SCHEMA(k,j) = 0

The "0" elements serve to aid the system in diagnosing erroneous user queries. Also, these elements help to insure the integrity of the internal form representation. These concepts are detailed later in the discussion. For now, the reader can think of a "0" as indicating that the entity type pair is "somehow engaged" in a relationship.

Since an entity type cannot be related to itself (in the GERMS sense), each main diagonal element (SCHEMA(m,m)) is set to "0". Here, a "0" element indicates that the

entity type pair is "unrelatable." The presence of a blank matrix element (that is SCHEMA(i,j) = BLANK) indicates that the respective entity types of the GERMS schema are unrelated.

Figure 12.1a shows a simple schema graph which represents a small portion of the logical contents of census file STF1B. Figure 12.1b shows the internal form representation of this schema (the row/column labels are to aid the reader). Note that the internal form representation includes all logical geographic entity types regardless of the physical representation of the data. For example, the matrix represents both the entity type URBAN-TRACT* and the entity type TRACT despite the fact that these types would share a common physical representation.

Clarification of the SCHEMA matrix configuration is due here. Consider the "top most" relationship of the example schema graph -- that is SMSA COMPRISES URBAN-TRACT*. Now, in the internal form representation of this schema graph, SMSA and URBAN-TRACT* hold sequence numbers 3 and 6, respectively. Since COMPRISES is signified by "2", this integer value resides in the matrix element of the third row (SMSA) and the sixth column (URBAN-TRACT*) of the SCHEMA matrix (i.e. SCHEMA(3,6)).

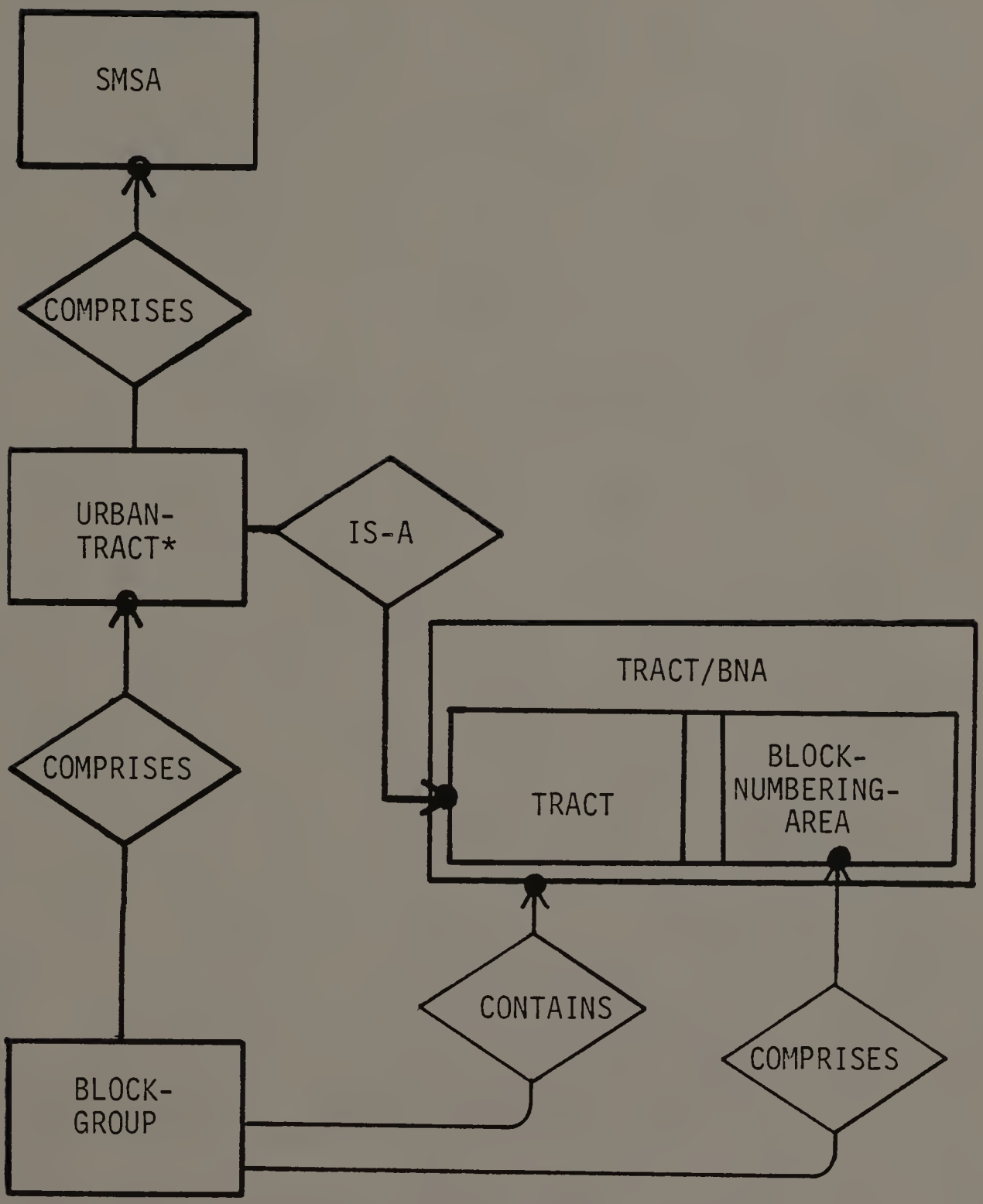


FIGURE 12.1a

A PORTION OF THE STF1B SCHEMA

	BLOCK-GROUP	BLOCK-NUMBERING-AREA	SMSA	TRACT	TRACT/BNA	URBAN-TRACT*
BLOCK-GROUP	0	0			0	0
BLOCK-NUMBERING-AREA	2	0		0	0	
SMSA			0			2
TRACT		0		0	0	3
TRACT/BNA	1	4		4	0	
URBAN-TRACT*	2		0	0		0

FIGURE 12.1b

THE INTERNAL FORM MATRIX REPRESENTATION

A "0" resides in the "transposition element," that is SCHEMA(6,3), to indicate that the URBAN-TRACT* - SMSA entity type pair is "engaged" in a relationship.

If the conceptual schema (including the implied relationships) is encoded into matrix form through the above rules, then the matrix captures the full information content of the schema structure. As such, the matrix is an equivalent schema representation to that of the user oriented GERMS graph. It is obvious that, unlike the GERMS graphical form, this matrix can be easily stored within the database machine and referenced by our GIS software. Indeed, this internal matrix, once created, comprises a major portion of our system's "knowledge" of the schema structure.

The problem of relating the entity type names used in system input and output (queries and responses) to the sequence numbers used in the internal form schema representation is easily solved through the use of a symbol table. The table merely lists each entity type name and its associated sequence number. This is the remaining portion of our system's "knowledge" of the schema. User reference (in a query) to an entity type name which is not contained in the symbol table indicates an error.

Once the internal form matrix and symbol table are in place, the system can easily verify the semantic correctness of user queries. Each entity type name, checked against the contents of the symbol table, verifies that "legal" type names are used. For example, continuing to use the same example schema, if the COUNTY entity type is used in a query, the query is rejected because COUNTY is not contained in the symbol table -- it is not a "legal" type name in this situation.

Beyond this, the system must investigate the accuracy of the relationships referenced in the query breakdown (e.g. "GET A BY B..." where "A " and "B" are "legal" entity type names). This is simple in that it must only be verified that a legitimate (CONTAINS or COMPRISES) relationship holds in the proper direction between the entity types.

In terms of our internal form schema representation this translates into verifying that a "1" or a "2" is stored in SCHEMA(i,j) where i relates to the superordinate entity type (i.e. preceeding the "BY" keyword), and j relates to the subordinate entity type (following the "BY" keyword). Thus, if a query of the form "GET BLOCK-NUMBERING-AREA BY BLOCK-GROUP" (this is an abbreviated query) is posed, it must be verified that a

"1" or a "2" resides at SCHEMA(2,1). If this is not the case, then the query is erroneous.

One use of the "0" elements can now be understood. Assume that a query is specified "backwards," say "GET BLOCK-GROUP BY BLOCK-NUMBERING-AREA." It seems reasonable that this would be a common user error. Upon detecting the "0" value at SCHEMA(1,2), the diagnostic message "RELATIONSHIP REFERENCED IN REVERSE ORDER" should be issued. Upon detecting a "BLANK" element, on the other hand, the diagnostic "NO SUCH RELATIONSHIP" is appropriate (this situation relates to the query "GET SMSA BY BLOCK NUMBERING AREA." Thus the "0" elements allow the system to better "understand" the nature of user errors.

Creating the internal form: data definition language. We have discussed how, once stored, the system's internal "knowledge" of the conceptual schema aids the system in "comprehending" user requests. The GERMS data model can thusly serve as the system's interface with the human user. We have not discussed how the GIS is "fitted" with this knowledge, however. This is the topic of the present and the following subsections.

For the time being our discussion will ignore two complexities. The first of these has to do with the implicit relationships of the GERMS schema. Recall that when the schema graph is employed these relationships are "derived in the user's mind." The second complexity ignored here involves the details of "linking" our internal form representation with the physical database configuration (i.e. record types and directories). Obviously, this linkage is important since we must eventually form a "bridge" between the logical and the physical domains.

It is assumed that the GERMS schema graph exists prior to the creation of the internal schema representation. Thus, the question addressed here is: Given the schema graphical representation, how can the details of the schema be loaded into the information system? The schema graph cannot be interpreted by the computer, so some intermediate representation must be found. This intermediate representation must obviously be an equivalent representation. In virtually all database management applications this problem is solved through use of a data definition language. Indeed, this is the tack taken here.

A data definition language is simply the language of a data modeling formalism. This language provides a means of describing the structures and constraints represented in the schema graph. The graphical form and the lingual form of a formalism must provide equivalent schema representations. The data definition language version of a schema is parsed and interpreted by the software of the DBMS. Once the schema representation is interpreted, construction of the internal form schema is not complex.

The GERMS language described in chapter 8 provides an obvious basis for a data definition language. The GERMS graphical and lingual representations are equivalent. Also, phrases of the GERMS language have a well structured and ordered syntax which can be easily parsed. Recall that there are only four different phrase forms each of which depicts a specific geographic relationship type. By including a GERMS phrase for each relationship of the schema graph, the entire information content of the conceptual schema is captured.

In the paragraphs that follow we describe the format of what we call the "schema definition program." The schema definition program represents our suggestion as to how the information system can be "fitted" with the "knowledge" of the conceptual schema. Interpretation of

the program by the GIS software would result in the formation of the internal form schema matrix and the symbol table. The software required to interpret a schema definition program has not been written as of yet, but the task would not be overly complex.

In the absence of this interpretive software the internal form components can be created by the database technician since the procedure is straight forward. Even in this situation where the internal form is created "by hand," we suggest that the schema definition program be written. This is because the program provides the database technician well structured documentation as to the internal form representation. It also serves as a stepwise guide to the required creation procedure.

The schema definition program consists of two sections. The first section, which is the ENTITY SECTION, enumerates the logical geographic entity types which are used in the conceptual schema. The basic purpose of this program section is to describe the structure of the symbol table and to define the size requirement of the schema matrix. The ENTITY SECTION also defines the link between the entity types of the conceptual schema and the record types of the physical database (this is discussed in greater detail later).

The remaining SCHEMA SECTION of the schema definition program consists of the GERMS language phrases which describe the structure of the conceptual schema. With this information the elements of the schema matrix can be "filled in" to represent the geographic entity-geographic relationship structure.

Figure 12.2 shows the schema definition program which corresponds to the simple schema graph of figure 12.1a (above). In the ENTITY SECTION of the program, the "ENTITY-TYPES ARE" phrase indicates that the schema involves six distinct entity types. Thus a 6 x 6 schema matrix is required for the application. Note that these six objects are logical entities; they do not necessarily represent six distinct record types in the physical database. The "RECORD-TYPES ARE" phrase states that the application will use three different record types in the physical structure.

The "NAMES ARE" phrase lists the six names of the logical entity types. "END NAMES ARE" indicates the completion of the list. The entity type names of this list are the basis of the symbol table. Unless stated otherwise via the "RECTYPE =" clause, the physical record type used to represent a logical entity type holds the same name as does the logical type itself.

```
ENTITY SECTION
   ENTITY-TYPES ARE  6.
   RECORD-TYPES ARE  3.
   NAMES ARE  BLOCK-GROUP;
              BLOCK-NUMBERING-AREA, RECTYPE = TRACT/BNA;
              SMSA;
              TRACT,                RECTYPE = TRACT/BNA;
              TRACT/BNA;
              URBAN-TRACT*,         RECTYPE = TRACT/BNA;
   END NAMES ARE.

SCHEMA SECTION
SMSA COMPRISES URBAN-TRACT*.
URBAN-TRACT* IS-A TRACT.
TRACT/BNA IS EITHER TRACT OR BLOCK-NUMBERING-AREA.
URBAN-TRACT* COMPRISES BLOCK-GROUP.
TRACT/BNA CONTAINS BLOCK-GROUP.
BLOCK-NUMBERING-AREA COMPRISES BLOCK-GROUP.
```

FIGURE 12.2

A SIMPLE SCHEMA DEFINITION PROGRAM

In the example program (figure 12.2) we see that the three entity types; BLOCK-NUMBERING-AREA, TRACT, and URBAN-TRACT*; will not hold their own record types in the physical database. Rather, the instances of these different types will all be represented as instances of the TRACT/BNA physical record type (specified by "RECTYPE = TRACT/BNA").

The interpretation of the SCHEMA SECTION is straightforward. Each phrase relates to a unique relationship (association) of the schema graph. The entity types in a phrase indicate, via the symbol table, the appropriate rows and columns of the schema matrix to which the phrase relates. The relationship type used in the phrase defines, according to the aforementioned rules, the values that the schema matrix should take on. When this process is carried out for a SCHEMA SECTION phrase, we say that the phrase is "executed."

Figure 12.3 provides an example of this process. Here, we see how the system would execute the GERMS phrase SMSA COMPRISES URBAN-TRACT*. First, with regard to the entity types SMSA and URBAN-TRACT*, we see that the symbol table directs us to the third row and to the sixth column of the schema matrix, respectively. This row-column pair defines a unique element of the matrix; namely,

SCHEMA(3,6). The GERMS phrase involves a COMPRISES relationship, so SCHEMA(3,6) is set to the value "2". Also, SCHEMA(6,3) is set to the value "0" to indicate that the respective logical entity type pair is "involved" in a relationship.

Obviously, a non-blank (including "0") value should not be reset during this process. Reference to a SCHEMA element which holds such a value indicates an error in the SCHEMA SECTION of the schema definition program. This illustrates another use of the "0" elements, including the main diagonal elements: they help to insure the integrity of the internal form representation.

To summarize the content of this subsection, we have described the structure of a schema definition program which is based on the GERMS data definition language. The discussion here is ignorant of two issues: implicit geographic relationships, and the details of linking the physical structure with the logical structure. These concerns are discussed in the remainder of this chapter.

Again, we suggest that the schema definition program be written even if its interpretation and execution is not automated. This is because the program listing provides stepwise detail and documentation to guide the database technician through a "by hand" creation of the GIS internal form.

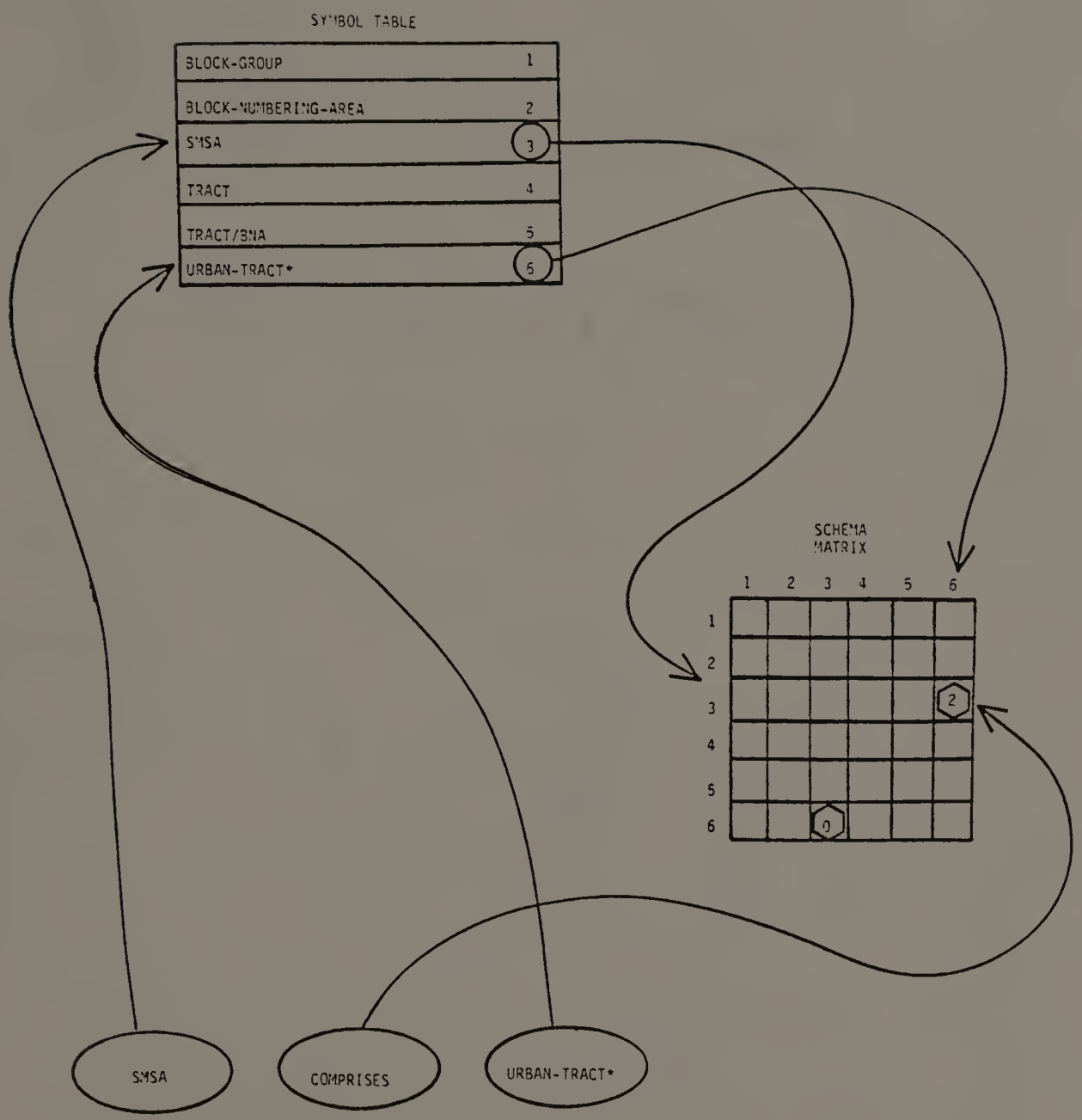


FIGURE 12.3

EXECUTING A PHRASE OF THE SCHEMA SECTION

Deriving implied relationships. In the two previous subsections we have discussed how a "knowledge" of the structure of the conceptual schema can be stored within our information system, and how this knowledge can be loaded into the system. In discussing each of these topics we have ignored the fact that a number of implicit relationships -- those which are not shown explicitly in the schema graph -- are inherent in the structure of the conceptual schema.

The implicit relationships are apparent from understanding the semantics of the explicit relationship structure; they are "obvious" to the human user who observes the GERMS schema graph. A graph showing all GERMS relationships would be too cluttered to be usable, so in terms of user effectiveness it makes little sense to detail the implied relationships in the structure of the schema graph.

Since implied geographic relationships are "obvious" to the data user, (s)he should be allowed to make references to these relationships in requesting a query breakdown. For example, if the schema graph shows (explicitly) that STATE COMPRISES COUNTY, and that COUNTY CONTAINS TRACT, then it makes sense for the user to request information in terms of "STATE BY TRACT..." This

request should be allowed despite the fact that the graph does not detail explicitly that STATE CONTAINS TRACT. Thus, the system should "understand" any user references to geographic relationships as long as they do not contradict the meaning presented by the schema graph.

If reference to a geographic relationship is to be "understood" and processed by the system, it is required that (1) the system "knows" of the relationship since the validity of each query is verified with the internal schema matrix, and (2) there exists a directory for the relationship within the physical database. In other words, the relationship must be fully represented within the GIS internal form.

The problem addressed here can be stated as follows: How can this "full knowledge" of the structure of the conceptual schema be gained by the information system? Solving this problem is made easier by the fact that we need only be concerned with implied COMPRISES and CONTAINS relationships here. This is because only these two relationship types can be directly referenced in a query.

One solution to this problem revolves around the creation of a set of rules through which the implicit relationships can be derived from the structure of the explicit conceptual schema. Such a set of rules is listed

below. These rules are based on following unbroken "relationship chains" (i.e. relationship type --> entity type --> relationship type...) through the network structure of the explicit schema.

Figure 12.4 provides a structural template for the rules. The template specifies three logical entity types represented by A, B, and C. Entity types B and C are connected by relationship Y which symbolizes a standard GERMS relationship (association) type. Relationship X, which is also a standard GERMS type, depicts the start of the chain.

Any number of entity type-relationship type pairs form an unbroken chain from relationship X to entity type B (these relationships may be any mix of the four standard types). The task at hand is to discover the nature of relationship Z which joins (implicitly) entity type A with entity type C. Note that the direction of the relationships shown in the template <u>is significant.</u>

As an example of how this template is used, assume that A represents SMSA, that B represents URBAN-TRACT*, and that C represents BLOCK-GROUP. Therefore, referring back to the earlier figure 12.1a, we see that both X and Y represent COMPRISES relationships. The problem addressed here is to uncover the nature of the relationship which

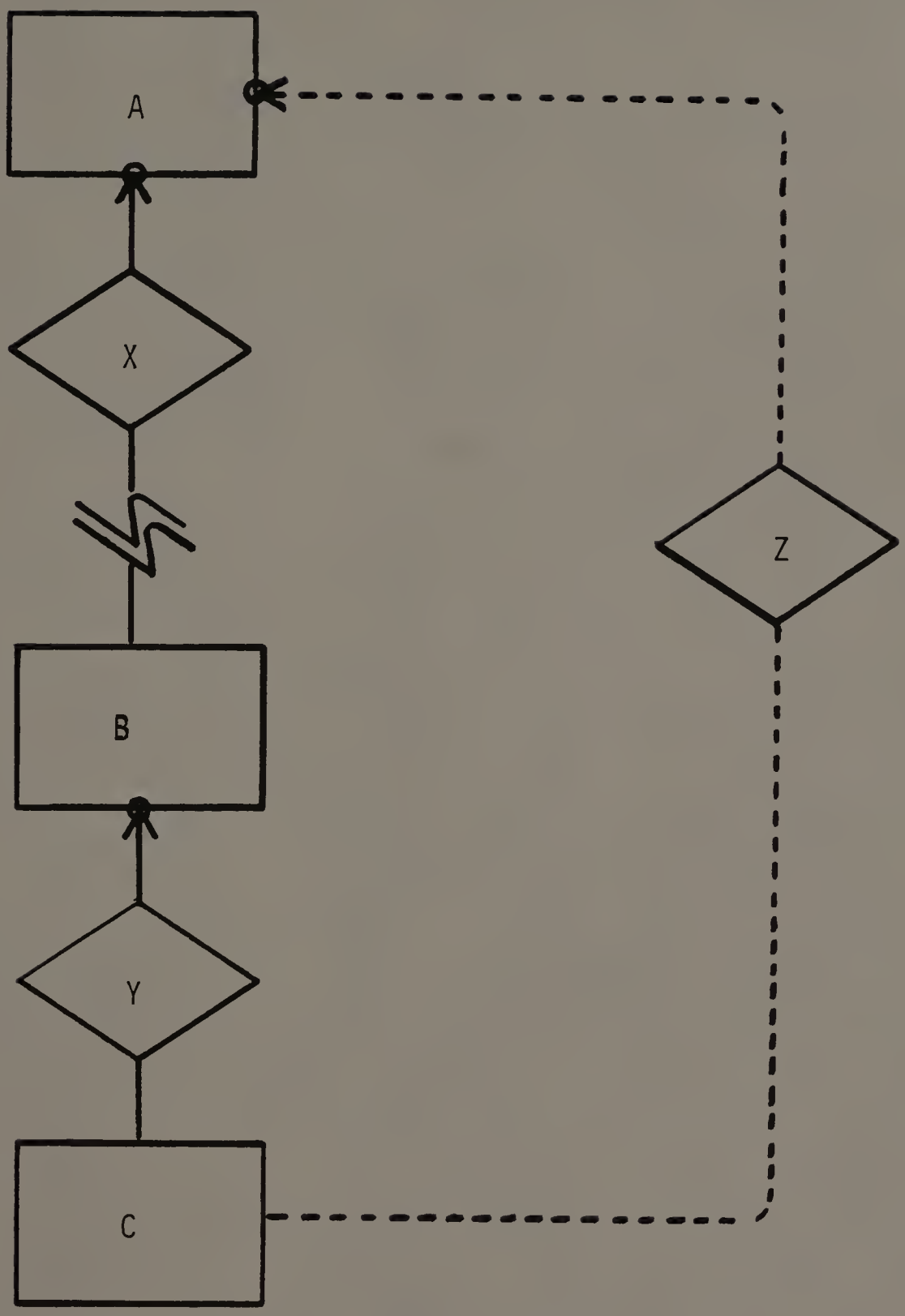


FIGURE 12.4

A STRUCTURAL TEMPLATE FOR DERIVING
IMPLIED RELATIONSHIPS

holds between SMSA and BLOCK-GROUP, that is relationship Z.

Our task is simplified when we realize that we only have to consider those chains which begin with (i.e. relationship X is) CONTAINS or COMPRISES. This is because it only "makes sense" to nest within the more specific, or "lower level" (vis. more general or "higher level"), entity types. The skeptical reader should verify this before continuing.

The rules for deriving the type of the relationship Z are stated using an IF-THEN-ELSE structure in the usual way. The rules are as follows:

```
IF   B CONTAINS C
THEN A CONTAINS C

IF   B COMPRISES C
THEN IF   all preceeding relationships are COMPRISES

     THEN A COMPRISES C

     ELSE A CONTAINS C

IF   C IS-A B (note the order here)
THEN A CONTAINS C

IF   B IS EITHER (OR) C
THEN A CONTAINS C
```

In processing the network structure of a GERMS schema graph each implicit or explicit CONTAINS or COMPRISES relationship represents the start of a chain. The process of deriving the implied relationships is therefore iterative in that one implied relationship may denote the presence of other implied relationships. In the case where an entity type in a chain has two relationships entering (pointing toward) it, the path branches to form two separate chains.

With regard to the row-column structure of the internal form schema matrix, the above rules can be implemented by moving from row to column to row of the matrix. For example, if the start of a chain is defined by the presence of a "2" (COMPRISES) in SCHEMA(i,j), then a "next" element of this chain would reside in the jth row of the matrix. Let us assume that such an element is found at SCHEMA(j,k). In this case, if the chain continues, then the next element resides in the kth row of the matrix; and so on until a row contains only blank or "0" valued elements.

Using this basic technique an algorithm can be employed to automatically "fill in" the implied relationships of the schema graph. With such an algorithm, which can be implemented using a stack

structure, the GIS has the capability of deriving a "complete knowledge" of the schema from its partial knowledge of the explicit relationships of the schema graph (as described by the schema definition program).

A less complex alternative, which we recommend, is for the database technician to derive the implied relationships by hand according to the above simple rules. The additional (implied) relationships, when included in the SCHEMA SECTION of the schema definition program, would result in the fitting of the system with a full knowledge of the schema structure.

Linking the Physical Database to the Logical Database

So far in this chapter we have discussed how a full knowledge of the conceptual schema can be created and stored within our geographic information system. A remaining issue deals with how this internal representation of the logical data structure, which we call the logical database, is linked to the physical data structure, or physical database.

The reason that these strucures must be linked is that the GIS is queried in terms of the objects of the logical database, but the physical database is consulted in the formation of responses to these queries. Thus the system must know which objects of the physical database relate to which objects of the logical database. The nature of this linkage is the topic of this section.

We refer to this issue as "intra-system mapping" because, when our geographic information system (GIS) is viewed as a single entity, said mapping is completely internal to the system. Continuing along this line, if the logical structure and the physical structure are considered as component parts of the GIS, then the mapping relates to the interconnections between (or coupling of) these components. When witnessed from outside the system, as by the data user, these interconnections are unobservable. Thus the overall GIS is an example of Ashby's "black box" (see [ASHB64]).

The inputs to this black box are the user's queries; the outputs are query responses. The internal "mechanism" of the black box is not "seen" by the user, and this is as it should be, for an understanding of geographic information does not presuppose an understanding of data structures and data storage techniques. This thought,

which arises from taking a more or less "cybernetic view" of our GIS returns us to the DBMS principle of data independence -- the user should be insulated from the complexities of the data organization. Indeed, intra-system mapping is how this data independence is achieved.

Stated in simple terms, the problem addressed by the mapping is twofold. Specifically:

1. Given the name of a logical entity type, the system must be able to direct itself to the appropriate record instances of the physical database

2. Given a reference to a relationship through an entity type pair (e.g. A BY B), the system must be able to direct itself to the appropriate relationship directory of the physical database (recall that any entity type pair denotes at most one logical relationship)

The first task is handled easily by expanding the aforementioned symbol table. Note that up to this point our symbol table has been described as a simple list of entity type names along with the respective entity type sequence numbers. This structure serves to associate the

logical entity type names with the rows and columns of the internal form schema matrix. Consider now that this same symbol table can be expanded so that each row also specifies the record index that relates to the entity type name.

Recall that each entity type name relates to exactly one record type, but each record type may be related to more than one entity type name. For example, the entity types TRACT and TRACT/BNA are each related to the record type TRACT/BNA. This single record type is therefore related to both entity types.

In the case where a number of entity types share a common record type, as in the stated example, each of the entity types has its own dedicated index through which the record instances are accessed. In this way we can support logical redundancy in the schema without the need for physical redundancy of database records. Now, if the symbol table holds pointers which are directed to the appropriate record indexes, then the system can always determine which record instances are related to which logical entity types.

Figure 12.5 illustrates this technique. The diagram shows a portion of the physical configuration of the schema of figure 12.1a (above). Here the four logical

entity types -- BLOCK-NUMBERING-AREA, TRACT, TRACT/BNA, URBAN-TRACT* -- share the single physical record type TRACT/BNA. User reference to a logical entity type name denotes, through the symbol table, reference to a particular record index. Each index further directs us to the appropriate physical record instances. The double arrow (>>) symbol denotes a set of pointers (vis. a single pointer).

The reader may be wondering how, at the time of database population, the system knows to relate these four different index structures (BLOCK-NUMBERING-AREA, TRACT, TRACT/BNA, URBAN-TRACT*) to the common TRACT/BNA record type. The answer is simple. The information required to define this configuration is contained in the ENTITY SECTION of the schema definition program that was discussed earlier (refer back to figure 12.2). In this program the logical entity types BLOCK-NUMBERING-AREA, TRACT, and URBAN-TRACT* are linked to the record type through use of "RECTYPE = TRACT/BNA." The TRACT/BNA logical entity type is linked to its synonym physical record type by the absence of this "RECTYPE =" clause.

As for the remaining part of our task at hand, linking a reference to a logical geographic relationship to its physical directory representation, we will see that

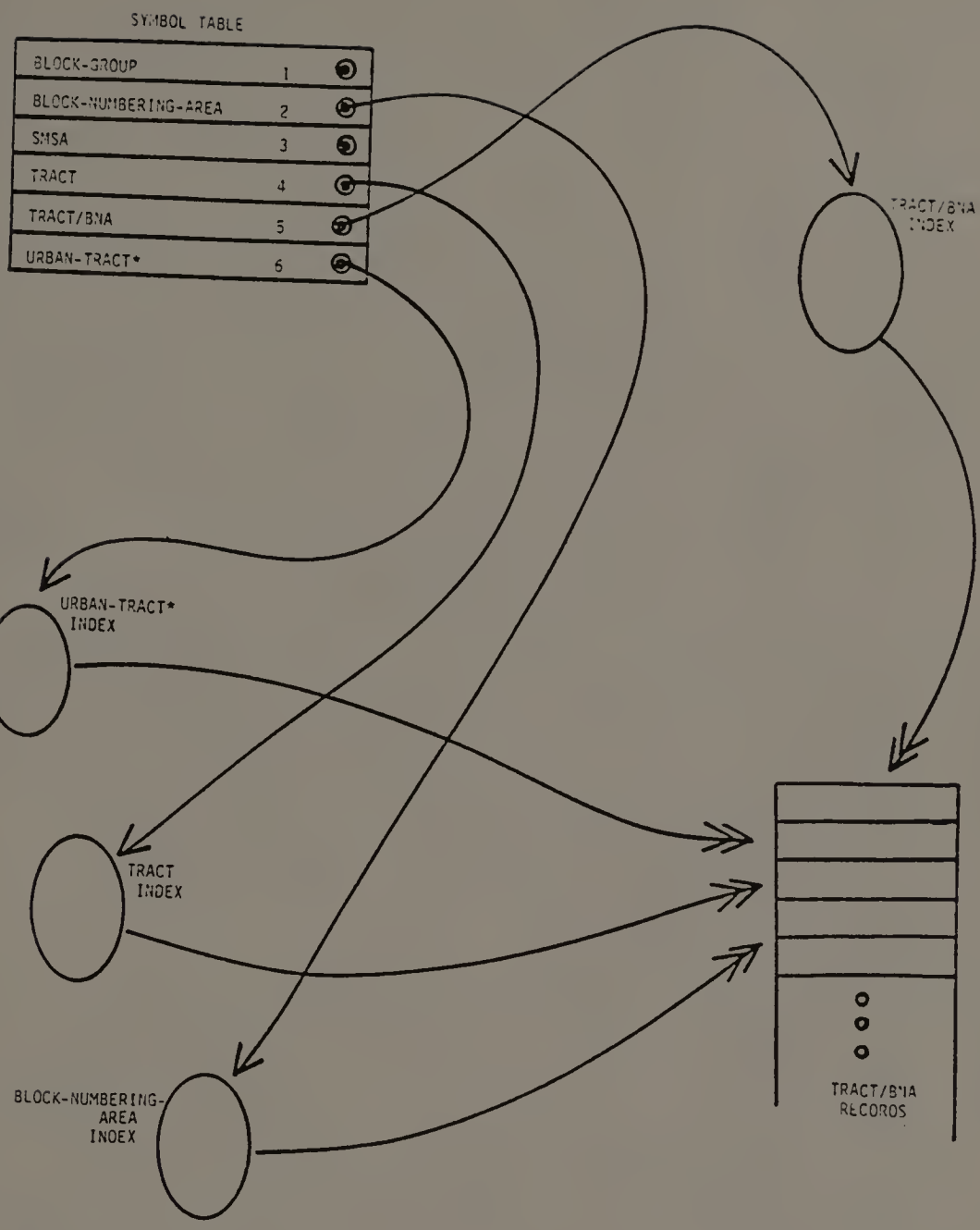


FIGURE 12.5

RELATING ENTITY TYPES TO RECORDS

it is not complicated. The simple solution capitalizes on the fact that each logical geographic relationship is associated with exactly one logical entity type pair. Indeed, this fact allows for the use of our aforementioned two-dimensional internal form schema matrix.

Reference within a query to a relationship is made in terms of an entity type pair (e.g. A BY B), so we need only relate each entity type pair to its respective relationship directory in the physical database. The pair-wise nature of the relationship allows for us to use a two-dimensional "pointer matrix" for this purpose. Each non-blank element of this matrix is a pointer which is directed towards a relationship directory -- which one being determined by the row-column position (entity type pair) of the element. We will call this matrix, which is the final component of the logical database, POINTER.

In figure 12.6 we show how the pointer matrix is used in handling the query "GET SMSA BY URBAN-TRACT* BY BLOCK-GROUP" (this query is based on the schema of figure 12.1a). In describing the illustration we will refer to the pointer residing in the ith row and the jth column of the matrix as POINTER(i,j). The problem here is to locate in the physical database two relationship directories: SMSA COMPRISES URBAN-TRACT* and URBAN-TRACT* COMPRISES BLOCK-GROUP.

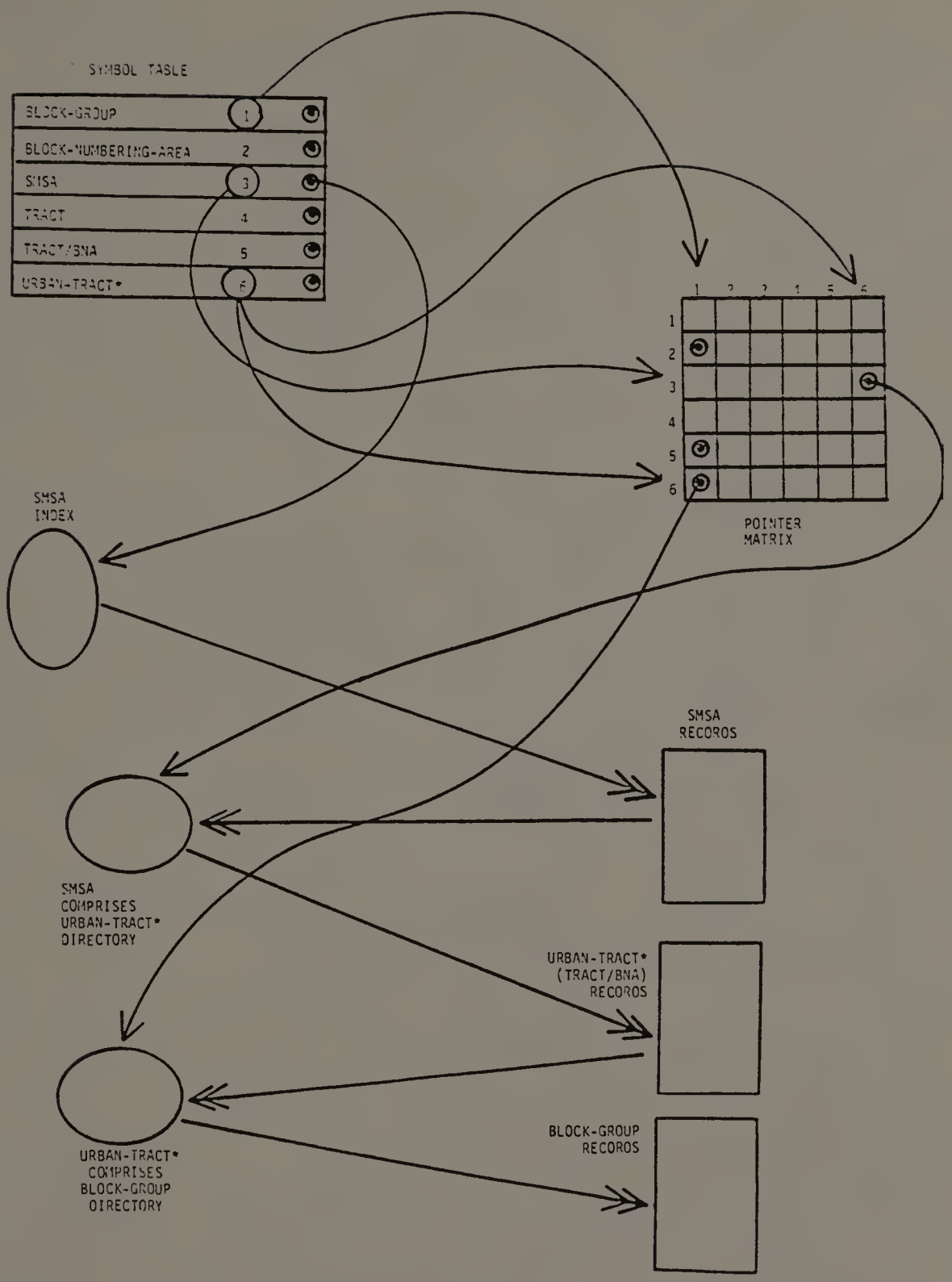


FIGURE 12.6

USING THE POINTER MATRIX

As defined by the symbol table, the SMSA logical type holds sequence number 3 while the URBAN-TRACT* logical type holds sequence number 6. The relationship directory associated with this entity type pair is therefore pointed to by matrix element POINTER(3,6). Similarly, URBAN-TRACT* holds sequence number 6 while BLOCK-GROUP holds sequence number 1, so POINTER(6,1) is directed towards the relationship directory for this entity type pair. Note that this phase of query processing would not be carried out until the semantic validity of the query breakdown was checked by the system (i.e. verifying that a "1" or a "2" resides at both SCHEMA(3,6) and SCHEMA(6,1)).

Recapitulation

The system's knowledge of the conceptual schema is stored in two main components: the schema matrix and the symbol table. Each element of the schema matrix specifies the presence or absence, direction, and type of a relationship which is held between a pair of logical entity types. The specific entity type pair is defined by the row and column in which the matrix element resides.

The symbol table associates the entity type names with the rows and columns of the schema matrix.

Using these two components the GIS can verify the semantic validity of any query which is posed to the system. The "legality" of the entity type names is checked against the entries of the symbol table. The correctness of the query breakdown request is checked by inspecting the values of the appropriate elements of the schema matrix.

The symbol table is also used to link the entity types of the logical database to the records of the physical database. Each row of the symbol table holds a pointer which is directed towards the respective record index. The entries of each index are in turn directed to the appropriate data records.

The relationship directories are located by the system through the use of a third logical database component, the pointer matrix. Each row-column combination of the matrix denotes an entity type pair. If a proper relationship holds between an entity type pair, then the respective element of the pointer matrix is directed toward the directory for this relationship.

The system's knowledge of the conceptual schema is loaded by means of the GERMS data definition language. A schema definition program defines the structure and content of the schema matrix and the symbol table. The implicit relationships of the schema can be derived by the system, or alternatively they can be derived by hand and entered as part of the schema definition program.

Looking ahead. The capabilities of our GIS which have been discussed to this point relate to a very basic system. We feel that the query structure is both flexible and powerful excepting one fact. Namely, geographic area primary keys must be employed in detailing which data are to be retrieved from the database. This obviously limits the suitability of the system for many applications. The ability of the system to handle associative queries -- data retrieval requests which are based on non-key attribute values -- would greatly widen the scope of system applicability.

Also, only very basal integrity checking mechanisms have been discussed. The technique with which the system verifies the integrity of a query has been described, but a means for maintaining the integrity of the overall database has not been detailed. Database integrity is an

important consideration in any information system for obvious reasons.

In the following chapter we discuss the issues that surround the addition of these capabilities to our GIS. The discussion has been delayed until now in an attempt to make the above system description more manageable. The system described up to this point can be thought of as being the "basic GIS." The next chapter therefore introduces some system enhancements, or "amenities."

In addition to the above noted amenities, the subsequent chapter includes a discussion of two other possible enhancements. Specifically, these are a database merging facility and a user view facility. The addition of a merging facility allows for the logical data structure to be used as a "guide" to direct the merging of two or more GERMS structured databases.

The user view facility has to do with the ability of the GIS to support "sub-views" of the database conceptual schema. Different data applications require consideration of different subsets of the logical database contents. When user views are employed, the user need only deal with that portion of the logical database structure which is relevant to her or his specific application.

Appendix C provides an illustration of a partial emulation of a "GERMS front-ended" database management system. This appendix describes the logical database structure and the physical database structure of an actual GERMS implementation. In addition, sample software which provides the system with some of the capabilities discussed above and in the next chapter are included.

C H A P T E R X I I I

SOME SYSTEM AMENITIES

In this chapter we discuss briefly a number of amenities which can be used to enhance our geographic information system (GIS). The discussion relies heavily on a thorough understanding of the "basic" system as described in the previous chapters.

Associative Data Retrieval

To this point in the discussion we have dealt with two general styles of query. The first style relates to the following request as phrased in free form English: "Retrieve this list of attributes for the instances of this entity type which hold these key value(s)." This request is posed to our GIS as

```
GET attlist FOR etype keylist
```

(we will ignore query breakdown for the time being).

The second form of query relates to the following request: "Retrieve this list of attributes for all instances of this entity type." The GIS formal query is

GET attlist FOR etype ALL

For some applications these simple query forms may not be enough. Consider, for example, the situation where a user desires an answer to the following type of request: "Retrieve this list of attributes for the instances of this entity type whose properties satisfy these conditions (e.g. population > 10,000)." A query of this type is called an "associative query" -- it requests data retrieval on the basis of (non-key) attribute values rather than on the basis of record key or record position.

The user of an associative query does not know the key values of the records in which (s)he is interested. Using our present query forms, then, the only option available is to request ALL instances of the entity type and then select "by hand" those instances which satisfy the condition(s) in question. When an entity type has many instances, such a process is hardly trivial.

The GIS can be greatly enhanced by including the ability to handle associative queries. In keeping with the same basic format, we suggest that the following associative query template be used:

GET attlist FOR etype1 WHERE boolean [BY etype2] [[AND] BY etype3]...

Here, the terms "etype" and "attlist" hold the same meanings as before. The term "boolean" represents some statement of required condition phrased in terms of attribute names, constants, relational operators, logical connectors, and parentheses. Conceivably, "boolean" can be of any level of complexity. The "WHERE clause" replaces the previous "keylist" or "ALL" phrase; otherwise, the query format remains the same.

Inclusion of the ability to handle associative queries should not affect greatly the basic physical structure of our GIS. Recall from chapter 11 that query processing involves the use of a "chain" of index and directory entries (i.e. index --> directory --> index...). Once the appropriately keyed record of the highest (nesting) level entity is located, all relevant "children" of the record can be easily retrieved.

With this in mind we realize that the processing of an associative query can be, for the most part, the same; it is only the way in which the chain is "entered" that need be changed. In other words, we must only implement some means of locating a record according to its attribute values.

There are two obvious ways of achieving this ability which do not disrupt the current physical configuration. Each has its advantages and disadvantages. The first method involves the use of inverted lists (recall that our relationship directories are based on this structure). An inverted list can be thought of as a directory which is dedicated to a certain attribute of the records to which it points.

Each row of the directory relates to a specific attribute value. The entries in a row are pointers to the records which hold that value on the attribute (the rows are obviously of different lengths). Given any attribute value, then, the records which hold that value on the attribute, if any, can be located by inspecting the directory. When an inverted list relates to an attribute represented in a data set we say that the data set is "inverted on" the attribute.

Inverted lists are conducive to very fast processing of associative queries. With other methods, "false" record accesses, called "misses," can constitute a significant amount of wasted processing time. When using the inverted list technique only those records which satisfy the criteria at hand need be accessed. Thus, misses do not occur.

The major disadvantage of using inverted lists is that they require a large amount of additional storage space. Information which is contained in the records is duplicated in the inverted list [1]. It is possible to invert on only a portion of the attributes that are represented in a data set. This creates what is called a partially inverted data set which obviously requires less storage space than does a "fully" inverted set.

Note that the nature of the data discussed here requires that (1) the inverted lists be "partitioned by" record type, or (2) each record type have its own set of inverted lists. This is because, in our geopolitical statistical situation, each attribute applies to all record types, but data requests deal with a specific record (mapped from entity) type (i.e. ...FOR etype WHERE boolean...).

The second method which can be used for associative retrievals is the sequential scan method. Here, each record instance of the requested entity type is accessed and inspected for the appropriate attribute value(s). This technique requires no additional storage space, but it is time-wise inefficient since many false record accesses, or misses, must be made.

An associative query is a request for all instances which meet the specified criteria (i.e. boolean), so the system must inspect every instance of the record type when using the sequential scan method. This is because there is no way of determining, until the end-of-file is reached, when the "last" satisfactory instance has been accessed. It should be obvious that, when this technique is used, there should be expected many more "misses" than "hits."

To invert or not to invert. Inverted lists require large amounts of additional storage space, but they are time-wise efficient. Sequential scanning needs no extra storage space, but it is time-wise inefficient. What we suggest, then, is to invert on those attributes which are expected to be most often referenced in associative queries. In this way increased storage costs are incurred

only in those situations where the benefits of decreased access time will be realized.

For example, if it is decided that the attribute POPULATION will be frequently referenced in associative queries, then the attribute should be inverted on. If, on the other hand, it is expected that POPULATION will be referenced infrequently, then the attribute should not be inverted. In this latter case, the sequential scan technique will be implemented in the event that such a reference (e.g. "...WHERE POPULATION > 10000...") is made.

Obviously, inverted lists can be added or removed by the database technician as the application evolves. The decision as to whether or not to maintain a given inverted list should be based on the frequency of access by the attribute -- a value which can easily be maintained automatically by the system.

Note that the volatility of the inverted list structure of the database in no way affects the data records themselves. The lists point to the records, but the placement and the format of the records are "ignorant" of the fact that the inverted lists exist. Consequently, the addition or deletion of an inverted list does not require a major reorganization of the physical database.

Complex booleans. The boolean portion of the WHERE clause of a query may employ logical connectives (i.e. AND, OR, XOR). If the physical database is such that only a portion of the attributes are inverted, then it is possible for the boolean to be a conjunction or a disjunction which references both inverted and uninverted attributes. Depending on the logical connectives used and the mix of attribute types, some interesting situations can arise.

Consider a boolean statement which has two relational statements joined by a logical connective (e.g. WHITE-POP > 10000 AND BLACK-POP > 5000). In the event that both attributes are inverted, there is no need for false record retrievals; the exact set of records which meet this criterion can be selected using the inverted lists. Obviously, if neither of the attributes is inverted, all records must be retrieved and inspected. In the case where one attribute is inverted and the other is not, the nature of the processing depends on the logical connective used.

If the relational statements are joined by AND, then only those records which appear in the appropriate rows of the single inverted list need be accessed. This is because a record "hit" must match both relational

statements, so a record which is not represented in the single inverted list must be a miss. Note that some of the records accessed may still be misses, but the number of misses is much smaller than would occur using a sequential scan.

If, on the other hand, the relational statements are joined by OR or by XOR, then all records must be inspected regardless of the fact that one attribute is inverted -- the inverted list is of no use here since it does not indicate that some records must be misses.

The point to be made is this: when only a portion of the attributes of the data set are inverted, the inverted lists may be rendered totally useless by the structure of the associative queries which are posed to the system. This fact should be kept in mind when designing the associative access structure.

A Semantic Integrity Checker

In any database management application the maintenance of semantic integrity is an issue of major importance. Recall from chapter 3 that semantic integrity has to do with the meaning, or semantics, of the data set.

When a data set lacks semantic integrity, then, data values, or sets of or functions of data values, contradict the logical meaning of the data.

We will refer to declarative statements about semantic integrity as "semantic relations."

The absence of semantic integrity can arise for a number of reasons. In terms of our geopolitical statistical data environment, the problem would often stem from errors in the data collection or in the data coding process.

The database technician who is entrusted with the responsibility of maintaining semantic integrity is faced with two major problems: (1) (s)he must determine what semantic relations <u>should</u> hold among the objects of the data set, and (2) (s)he must somehow verify that these semantic relations <u>do</u> hold among the objects of the data set. This latter problem is often attacked by writing special purpose data retrieval programs which are based on the <u>specific</u> semantic relations of the <u>specific</u> application.

The semantically rich highly structured GERMS formalism can simplify the maintenance of semantic integrity of geopolitical statistical data. The meaning of geographic relationships among geographic entities

indicates what semantic relations should hold among the respective data values. Thus, the database conceptual schema details the semantic relations.

For example, let super(i) denote the array of attribute data values which relate to some ith geographic area of type SUPER. Let sub(i,j) denote the array of attribute data values which relate to some jth geographic area which lies within the boundaries of super(i). Sub(i,j) is an instance of entity type SUB.

Now, if SUPER CONTAINS SUB, then the following semantic relation should hold:

$$\text{super}(i) >= \sum_{j} \text{sub}(i,j);\ \forall i$$

Thus, the aggregation of the attributes of the subordinate area records should be less than or equal to the attributes of their superordinate area records for each relationship instance.

Similarly, if SUPER COMPRISES SUB, then the following semantic relation should obviously hold:

$$\text{super}(i) = \sum_{j} \text{sub}(i,j);\ \forall i$$

In other words, the aggregation of the attributes of the subordinate areas should exactly equal the attributes of their superordinate area for each relationship instance.

Now let general(i) denote the array of attribute data values which relate to some ith geographic area of type GENERAL. Also, let special(i) denote the array of attribute values which relate to the same ith geographic area instance of type SPECIAL where SPECIAL IS-A GENERAL. In this circumstance, the following semantic relation should hold:

special(i) = general(i)

Thus the values of the attributes of a geographic area are the same whether they relate to the special case view or to the general case view -- views of entities change, but properties of entities do not.

A similar semantic relation has to do with the EITHER-OR association. Let lower(i) denote the array of attribute values which relate to some ith geographic area of type LOWER. Let higher(i) denote the array of attribute values which relates to the same geographic area instance of type HIGHER where HIGHER IS EITHER (OR) LOWER.

In this final case the following semantic relation should hold:

$$higher(i) = lower(i)$$

The rationale for this is the same as in the previous case.

The above discussion shows how the meaning of a GERMS geographic relationship indicates the semantic relations which should hold among the respective database objects. As for verifying that the semantic relations do hold, the task is simplified by capitalizing on the structure of the physical database of our GERMS-based geographic information system (GIS).

First of all, the semantic relations which deal with the IS-A relationship and with the EITHER-OR association need not be verified. This is because, in the physical configuration of our GIS, geographic entities which depict these different views of an object (i.e. special-general or higher-lower) are represented by the same physical records. Thus, there is no way that these semantic relations (equality) can be violated; the respective attribute values must be equal.

Second of all, the semantic relations which deal with the CONTAINS and with the COMPRISES relationship can be verified by general purpose procedures. Since the GIS has a knowledge of the structure of the conceptual schema, the system can determine when the semantic relations should hold. Consequently, the GIS can be given the ability to verify its own semantic integrity.

The basic logic. Consider a semantic integrity checker which is a set of programs that can be executed upon the command of the database technician. The main program merely inspects each element of the conceptual schema internal form matrix. In the event that a CONTAINS relationship or a COMPRISES relationship is found, the execution of the appropriate semantic relation checking procedure is triggered.

For the purpose of this discussion we will refer to the procedure which checks the semantic relation of the CONTAINS relationship by the name of "CONTCHK" (CONTains CHecK). "COMPCHK" will refer to the procedure which verifies the semantic relation of the COMPRISES relationship.

The logic of the semantic integrity checker main program is shown in figure 13.1. This simple routine comprises two nested loops which perform a row major order scan of the square matrix SCHEMA. Whenever SCHEMA(I,J) = 1 (a CONTAINS relationship), the CONTCHK procedure is triggered. Whenever SCHEMA(I,J) = 2 (a COMPRISES relationship), the COMPCHK procedure is triggered. The program terminates when every element of the schema matrix has been inspected.

In executing one of the checking procedures, every instance of the relationship in question must be considered. This demands that the system be able to locate each superordinate (in the particular relationship) record along with the related subordinate data records.

The index for the superordinate entity type is pointed to by the ith row of the symbol table (contained in the logical database). Once the index is located, the records may be easily accessed. The appropriate relationship directory is located by following the pointer which resides at POINTER(I,J). This directory allows for the system to locate those subordinate records which are related to a particular superordinate record.

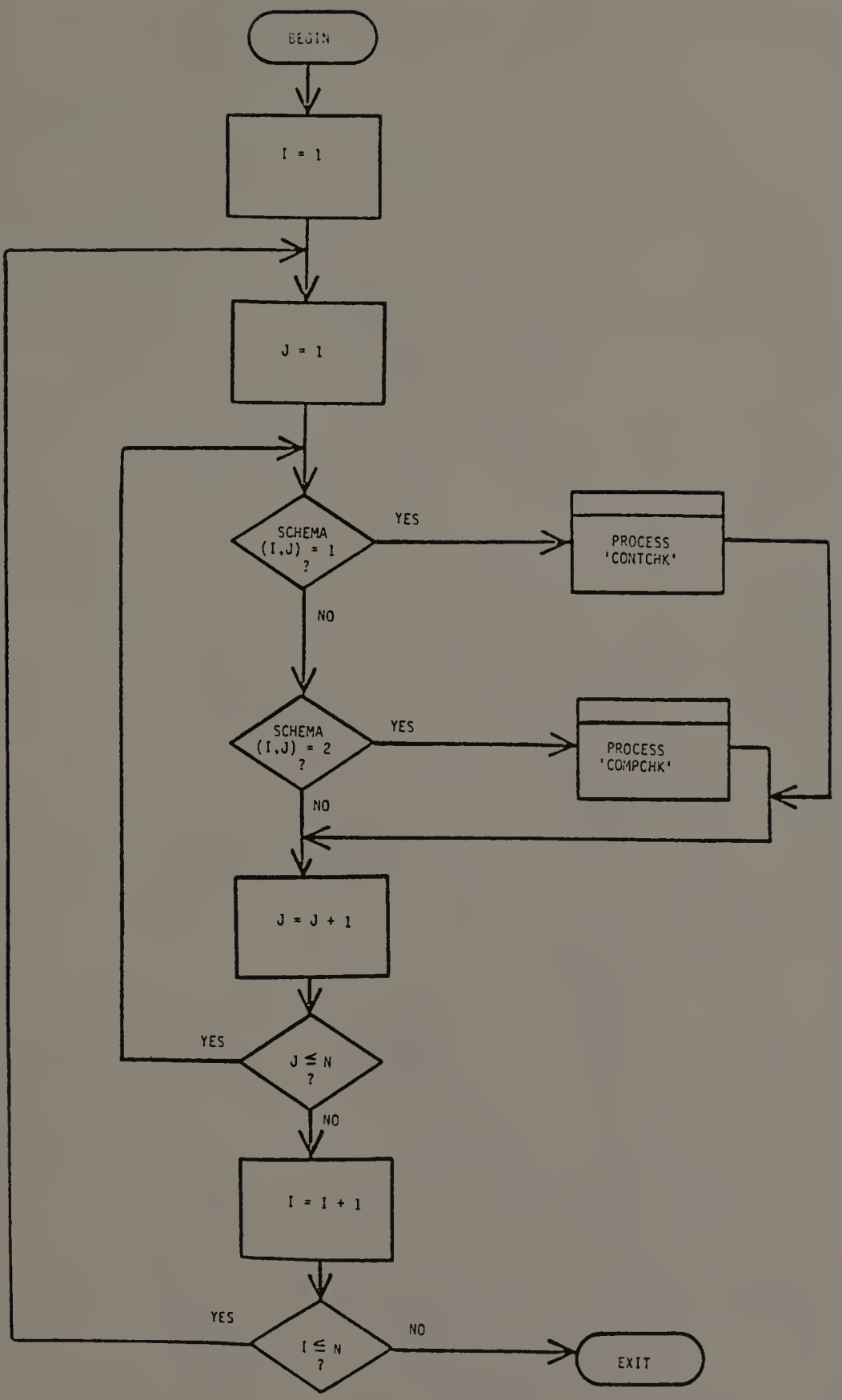


FIGURE 13.1

THE SEMANTIC INTEGRITY CHECKER MAIN ROUTINE

Figure 13.2 illustrates the logic of the procedure CONTCHK. Here, the variable AGGREGAT refers to a vector of values. Each element of the vector relates to a particular attribute which is carried in the data records. Any operation on this variable is meant to denote the operation on each element of the vector.

The logic of the procedure is straightforward:

> For each entry of the superordinate index, the data record is accessed. Then, all related subordinate records are accessed and their attribute values aggregated. If each aggregate value is less than or equal to the respective attribute value of the superordinate record, then the next relationship instance is checked. Otherwise, the execution of an error procedure is triggered.

COMPCHK uses the same logic with the exception that the aggregated values are checked for equality with the superordinate data values.

The error procedure indicates to the database technician the superordinate entity instance and the relationship in which the error occurs. For example, an error report could be of the form "COUNTY 'ABC' IN COUNTY CONTAINS TRACT/BNA." Also, the attribute names and the data values which are found to be in error are displayed.

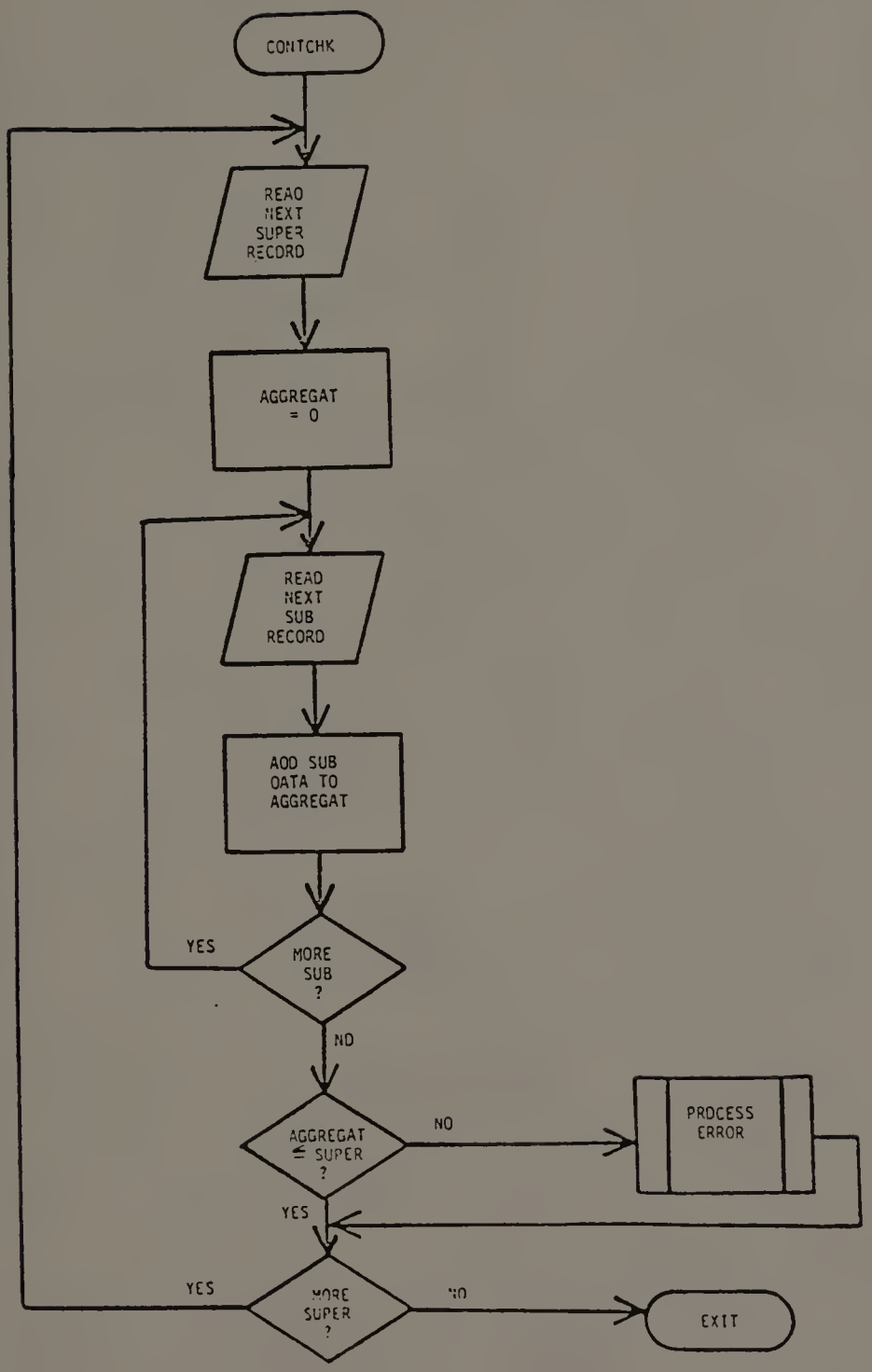


FIGURE 13.2

PROCEDURE CONTCHK

We recognize that the execution of these integrity checking programs would be expensive. It should be kept in mind, however, that the data in this application is static, so the checking need only be invoked once following database population, or after a rare database modification. The decision of whether to implement this semantic integrity checker should be based on the probability of and the costs associated with holding erroneous data values.

A Database Merging Facility

An attribute of the geopolitical statistical data environment is that there are many opportunities for the merging of databases. Whenever different properties denote assertions about common real world entities we observe the potential for the existence of this information within a common structure.

When geographic oriented data are collected, then, the data (measurements) are combinable provided that the geographic partitionings of the studies are compatible. By merging geographic based statistical data of differing orientations (e.g. sociodemographic, environmental,

economic) we open doors to a number of correlational and predictive research studies -- doors which would be otherwise closed.

The merging of conceptually modeled database is not well understood. Indeed, merging strategies for databases and database views has been cited [LUM78] as an area which deserves further research effort. It is our belief that the indigence extends to the geopolitical statistical data management environment.

In comparing the GERMS conceptual schemas of two databases, we are provided with a structure for database merging. When the databases hold different attributes for the same GERMS logical entities, the respective records can be combined. The GERMS schemas also indicate where new data can be generated from the initial merged data.

Whenever a database schema includes a COMPRISES relationship, and the subordinate (in the relationship) records are merged with the records of another database, the superordinate records can be expanded through the creation of new data objects. This new data is generated by aggregating the values of the merged data. The information required to direct the generation of the new data, that is, which subordinate records are related to which superordinate records, is contained in the COMPRISES

relationship directory of the original database. An example is provided in a later paragraph. Note that the structure of the relationship directory is untouched by this process; only the superordinate records are affected.

We suggest that the GIS can include a database merging facility. A set of programs aimed at combining the data of GERMS databases constitute this facility. In the event that new data can be generated from the merged data, as in the case described above, the process may be triggered by the following macro call:

GENERATE attlist FOR etype1 FROM etype2

Here, "attlist" denotes some list of record attribute names. "Etype1" and "etype2" relate to the superordinate entity type and to the subordinate entity type of a COMPRISES relationship, respectively.

By consulting the logical database, the system determines which relationship directory should be used to direct the data generation. The entity type pair "etype1, etype2" connotes, through the symbol table, a unique element of the POINTER matrix. This matrix element directs the system to the appropriate relationship

directory of the physical database.

An example. For the purpose of example we will refer to figure 13.3a and to figure 13.3b. Figure 13.3a shows a portion of the conceptual schema for census file STF1B. Figure 13.3b shows a portion of the conceptual schema for census file PL-94. Notice that PL-94 does not deal with SMSAs.

The nature of the statistical properties represented in these files is not important here. We will simply assume that these schemas relate to two databases which represent different properties of geographic areas. The attributes in the records of the first database will be called "STF1B-data." We will call the attributes in the records of the second database "PL-94-data."

Now, consider that we wish to merge the PL-94-data with the STF1B database. According to the conceptual schemas, the records of the COUNTY, TRACT/BNA, and BLOCK-GROUP/ENUMERATION-DISTRICT entity types can be combined. Note that, given the physical configuration of our GIS, this merging implies that the lower level TRACT, BLOCK-NUMBERING-AREA, BLOCK-GROUP, and ENUMERATION-DISTRICT are combined.

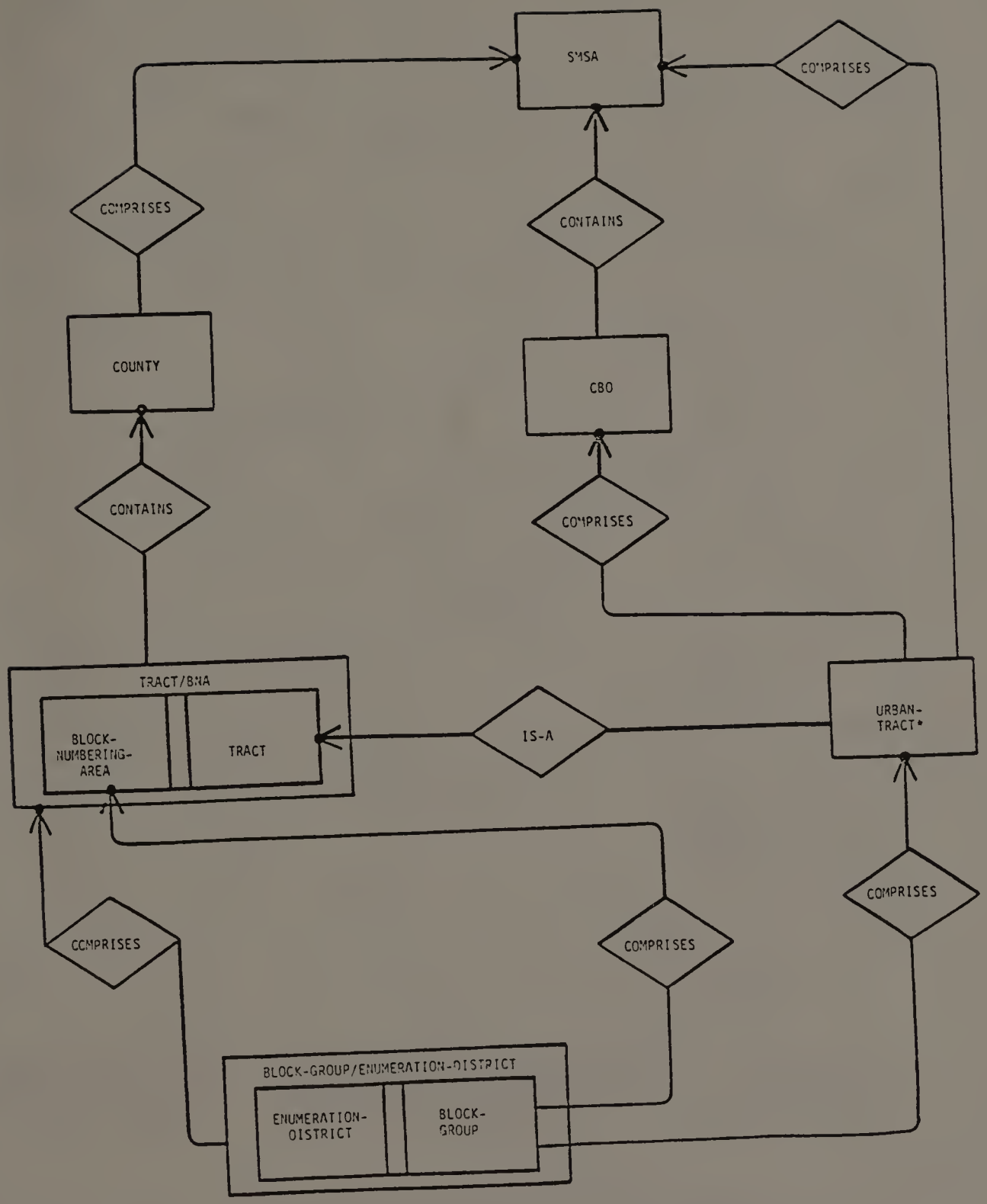


FIGURE 13.3a

A PORTION OF THE STF1B SCHEMA

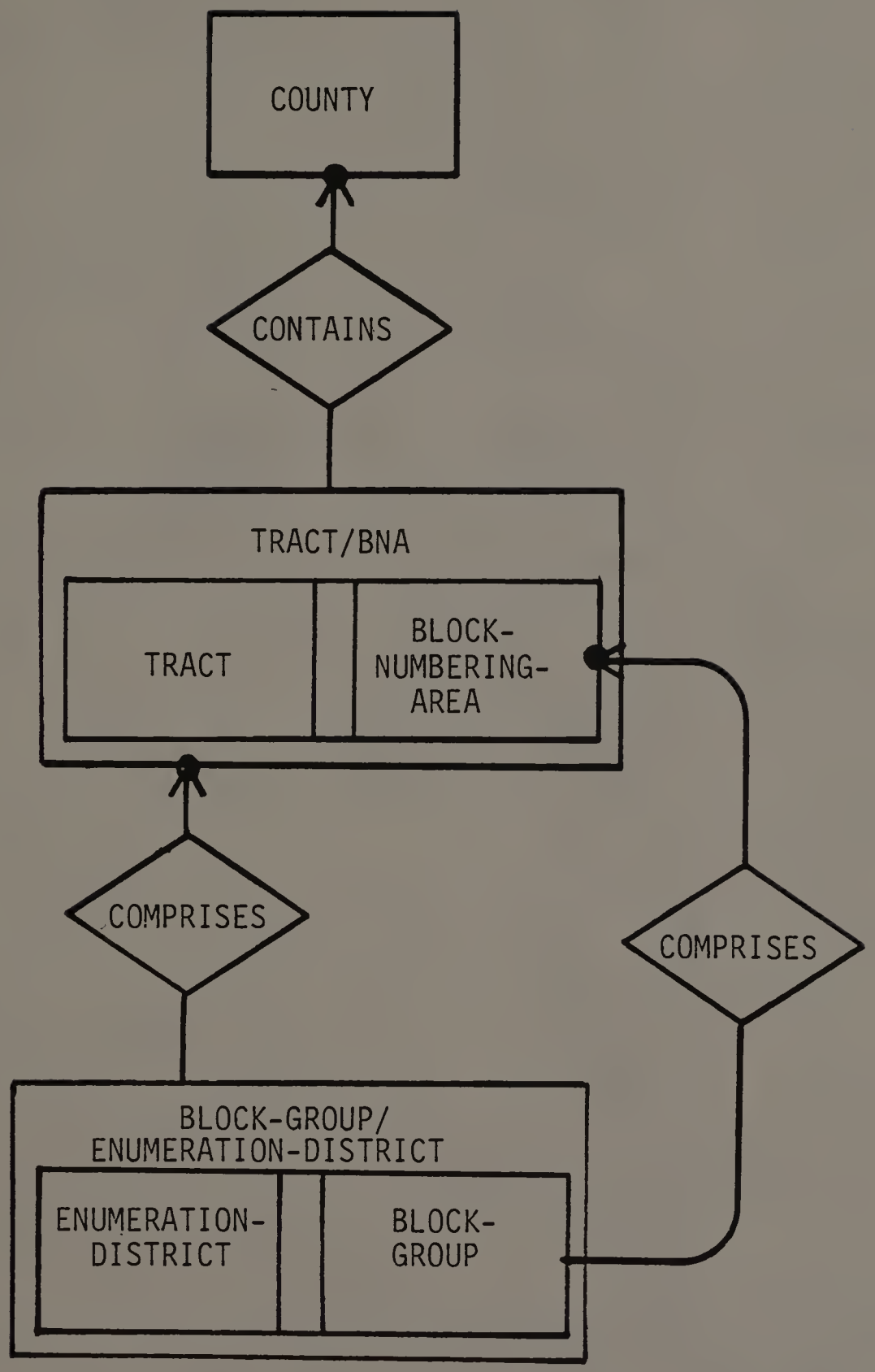


FIGURE 13.3b

A PORTION OF THE PL-94 SCHEMA

Since URBAN-TRACT* IS-A TRACT in the STF1B schema, and the URBAN-TRACT* records are indexed in the STF1B physical database, we now have PL-94-data for urban tracts*. Using the CENTRAL-BUSINESS-DISTRICT COMPRISES URBAN-TRACT* relationship directory as a guide, we can GENERATE PL-94-data FOR CENTRAL-BUSINESS-DISTRICT FROM URBAN-TRACT*. Also, using the relationship directory for SMSA COMPRISES COUNTY, we can GENERATE PL-94-data FOR SMSA FROM COUNTY. Note that, alternatively, we could GENERATE PL-94-data FOR SMSA FROM URBAN-TRACT*, but this would be more expensive since the schema indicates that there are more urban tracts* per SMSA than there are counties per SMSA.

The result of this data generation activity is that we can now request PL-94-data for any entity type of the STF1B schema. Again, in responding to these requests, the enlarged GIS relies on the same relationship directories as were used prior to the database merging. The GERMS relationship directories are untouched by the merging of GERMS databases; only the affected data records need be revised.

Figure 13.4 summarizes the above process. Each object in this diagram represents an entity type of the STF1B schema. The types whose data records are directly

merged with PL-94 records are represented by circles. The remaining entity types are represented by ellipses. Arrows indicate how data for these remaining areas is derived ("automatic" means that the PL-94-data exists automatically upon the merging of the database).

User Views

As the number of entities represented in a GERMS database increases, so does the complexity of the conceptual schema graph. For many user groups it may happen that the schema graph depicts many more entity types than are required for the application at hand.

Consider, for example, an application which is concerned exclusively with statistical geographic areas. In this case entity types such as SMSA, URBANIZED-AREA, and CENTRAL-BUSINESS-DISTRICT are relevant, but political areas such as COUNTY, MCD, and TOWNSHIP are of no interest. Consequently, a schema graph which depicts political areas and their respective relationships is more complicated than is required.

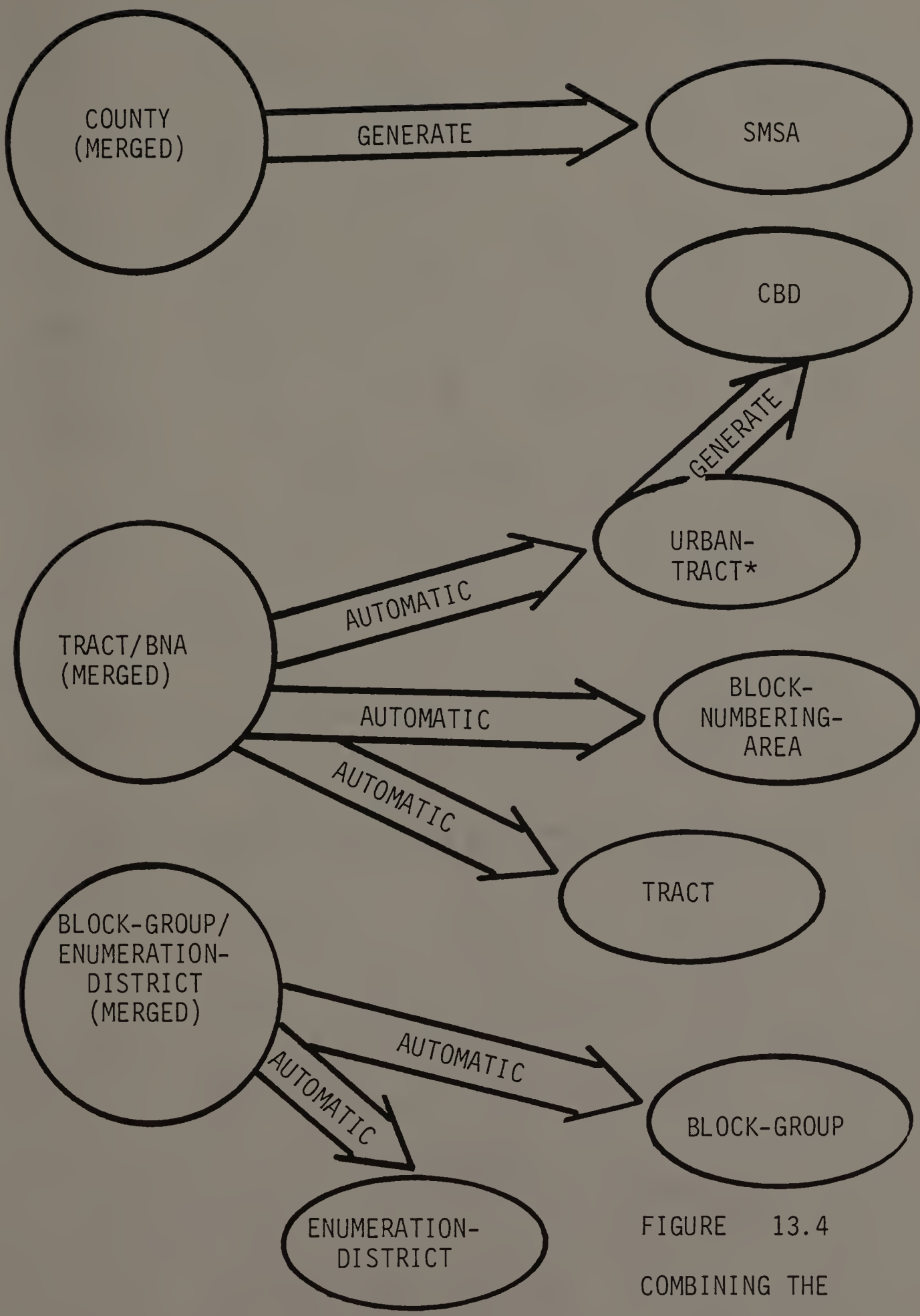


FIGURE 13.4

COMBINING THE DATABASES

A similar problem is faced by the users of common business databases. Here, we may have a single integrated database which serves the data needs of the entire business enterprise. A number of very different applications such as personnel, purchasing and production share the database.

These user applications are very different, so the single "global" logical data model (conceptual schema) is not well matched to the needs of any one application. Consequently, an application is often supplied with its own sub-model, or "subschema." Such sub-models are sometimes referred to as "user views." The ability of a data modeling formalism to support user views is called "relativism" (see chapter 6). Under this concept a user need only consider her/his particular user view when using the database. Thus, from the perspective of the user, it appears that the database exists to meet the needs of her/his application exclusively.

Returning to our geographic oriented GERMS database, consider the fact that a GERMS query deals only with those entities and relationships about which information is requested. Thus, if the user desires information in the form of "SMSA BY URBAN-TRACT*," there is no need to consider or to make reference to the fact that SMSA

COMPRISES COUNTY. Similarly, if information of the form "SMSA BY COUNTY" is desired, the user can ignore the fact that SMSA COMPRISES URBAN-TRACT*.

With this in mind we realize that a user can interact with the GIS without knowing the full logical contents of the database. Provided that each object referenced by the user agrees with the representation of the system internal form schema, the query can be interpreted properly by the system. Therefore, it is possible for a user to employ an incomplete schema graph and still make use of the information system.

Such a partial schema graph can be created to match the logical needs of a particular user application. This graph can depict the specific "user view" of the application [2]. As long as the subschema graph does not contradict the (global) conceptual schema, no adjustments need be made to the structure of the information system internal form. Thus, the GERMS formalism readily supports relativism.

The rationale behind employing subschema graphs is that a particular user community which requires only a subset of the total logical database objects can be insulated from needless complexity. Why, for example, should a user be forced to consider SMSAs, urbanized

areas, and central business districts (and the respective relationships) if her/his application is concerned with political geographic areas? By dealing with a "political oriented" subschema graph rather than with the global schema graph, the user's task is simplified. We suggest, then, that subschema graphs be employed whenever possible. Again, the use of these subschema graphs does not affect the database configuration in any way.

Figure 13.5 illustrates the use of subschema graphs. Here we see three distinct user groups each of which deals with a particular user application. Each user group views the common physical database "through the perspective" of its own (sub)schema graph. A user group need not be aware that the global view or that the other views exist.

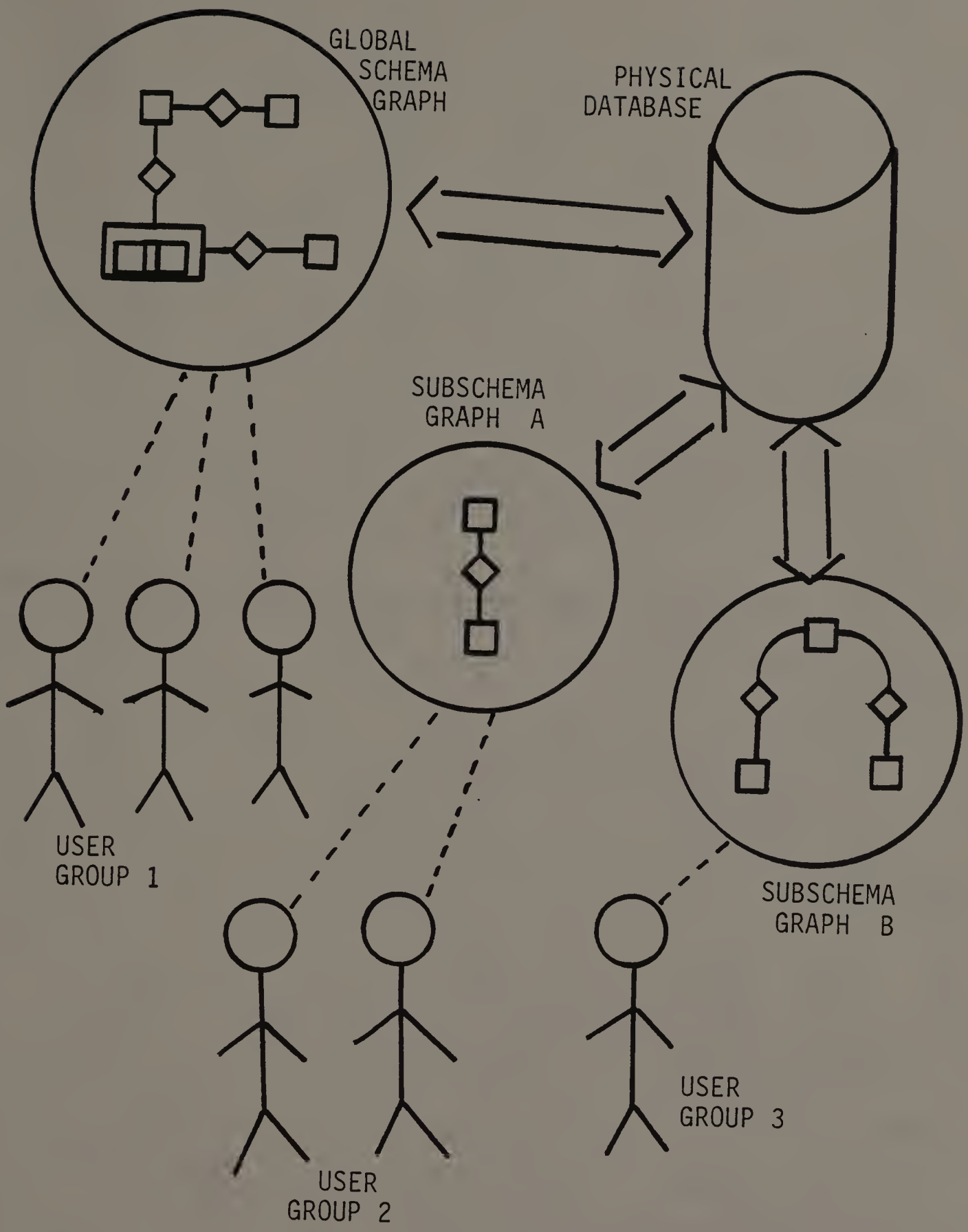


FIGURE 13.5

USER VIEWS

FOOTNOTES

[1] This is not true of our relationship directories. Recall that in forming these directories the equivalent information (ancestor key) is "stripped" from the source records before they are entered into the target structure.

[2] The reader is cautioned against confusing this use of the term "view" with the use employed previously. In the sections above we spoke of views of a geographic entity type (i.e. special-general, higher-lower). Here, we are referring to views of a data model.

C H A P T E R X I V

REVIEW AND CONCLUSIONS

The mission of this final chapter is twofold. First, the chapter provides a brief yet comprehensive review of the above chapters. Second, it investigates the consequence of our work.

The purpose of the review is to refresh our memories as to the major focal points of our work. Hopefully this retrospection will prove useful in assimilating the remaining portion of the chapter: the concluding remarks. These concluding remarks focus, for the most part, on the significance of our research described herein.

A Brief Review

In part I of this three-part work we provided a detailed (and often technical) description of conceptual data modeling and of data modeling formalisms. This part was aimed at providing a foundation for the research presented in the later sections; it was the background material. The discussion is extensive since we feel that meaningful research ideas deserve and require a solid base

on which they can be built. A significant portion of the text of this part focused, quite naturally, on data semantics -- the construct which underlies the concept of conceptual data modeling.

The discussion began by taking a very general, or "generic," look at data modeling and the issues related to such. A formalized notation and structural form was presented in the hopes that it would provide a communicative anchor for the more sophisticated and abstract concepts presented later in the discussion.

A second conceptual benchmark which was relied on was our existing understanding of programming and programming languages. We saw that many data modeling issues could be studied in terms of analogous, yet better understood, programming issues.

Lastly in our generic investigation of data modeling, we recognized the time-wise "generational development" of data modeling. We learned that the early first generation models (and modeling formalisms) were preoccupied with structure and with syntax, while the newer second generation models (formalisms) are more concerned with semantic richness.

The focus of our background discussion then turned towards the topic of data semantics as it relates to data modeling. We investigated, from a number of different viewpoints, the "meaning" of semantics. Indeed, we found that semantics is meaning.

Next, we studied data semantics from a strictly theoretical perspective. Following this theory section we considered the treatment of data semantics within the literature of database management.

In the succeeding three chapters, which comprise the remainder of part I, we were concerned with a number of fundamental concepts of data modeling. First, we investigated the nature of the objects and of the groups of objects (types) which form a data model. Next, we studied data abstraction -- a ubiquitous construct which is of two kinds. Specifically, these are aggregation abstraction and generalization abstraction.

Finally, in chapter 6, we focused on the issues of logical redundancy, relativism, and semantic integrity. These issues directed us into the realm of data model implementation, as they require that physical data be considered.

Beginning at part II, we redirected the discussion toward the specific application environment in which our interests lie -- that of geopolitical statistical data. First, we studied the physical structure of geopolitical statistical data sets. We learned that a number of conceptual anomalies arise when a hierarchical structure is taken to represent the logical structure of geographic based data.

Next, we looked at the true logical structure of such data. We found that, as can any data set, a geopolitical statistical data set can be modeled using only four basic kinds of objects. These are entities, relationships (associations), properties, and values.

The entities in this application are geographic areas. Relationships describe how the area entities overlap. Properties which hold values characterize the entities. These values which are held by the properties are statistical objects.

In accordance with our understanding of geographic logical structure we then developed a data modeling formalism which is dedicated to serving the special needs of the application at hand. We decided that, of the four basic object types, our data models should concentrate on only two: geographic entities and geographic

relationships. To that end, the formalism was named the Geographic Entity-Relationship Modeling System; hereunder simply the GERMS.

The GERMS has both a graphical and a lingual form. These forms provide data models which are "equivalent" in terms of information content. Each form serves its own purpose with respect to model usage.

As a close to part II we evaluated the GERMS formalism and considered it in the light of some of the concepts discussed in part I. Specifically, these concepts are data abstraction and logical redundancy. We learned that abstraction, as employed by a GERMS data model, does not match closely its standard "textbook" semantic. Also, we made the important discovery that a GERMS model is, by the nature of the formalism through which it is created, a logically redundant structure. This facet came into play repeatedly in later sections.

The above noted evaluation of the GERMS revealed that the formalism, as presented in part II, serves as (1) a medium through which the logical contents of the data set can be documented, and (2) a medium through which the logical contents of the data set can be discussed. In part III the usefulness of the GERMS was taken well beyond these two realms; the formalism was made an integrated part of a computer information system.

In part III, which comprises chapters 10 through 14, we developed and described the makeup of a GERMS-based geographic information system. Here, we use the term information system to indicate that said system has the ability to respond to ad hoc information requests. This system holds a "knowledge" of GERMS logical data structure and can therefore "comprehend" references to the GERMS data model.

First in this final part, we studied the format that a GERMS-based query should take on. This format should obviously be considerate of the need of the user. We decided that the query ought to have three parts: a focus part, a range part, and a (optional) breakdown part. These refer to properties, to entities, and to relationships of the data model, respectively.

Next, we developed a suitable physical database configuration. In this design, relationship data is separated from entity data. The relationship data is structured to form directories (inverted lists). Entity data holds a standard "flat" record type organization. Entity --> relationship --> entity traversal in the logical structure is emulated by moving from record type to directory to record type in the physical database.

In considering the physical database we were faced with the problem of dealing with the redundant nature of the logical structure. True, the two structures (logical, physical) are separate, but each logical object must be somehow represented in and linked to the physical database. This problem was easily solved through the use of additional indexes which allowed us to circumvent the need for employing physical data redundancy.

The subsequent issue which was addressed is that of logical database configuration. We defined the logical database as that part of the system which holds (1) the "knowledge" of the data model, and (2) the links which "connect" the logical model objects to their physical database counterparts.

While the variety of possible logical database configurations is virtually limitless, we chose one which comprises three basic parts. These are the symbol table, the schema matrix, and the pointer matrix. The schema matrix represents, unambiguously, the full inter-object structure of the GERMS data model using a single two-dimensional matrix. The symbol table holds the names of the logical entity types and couples these names with the appropriate portions of the schema matrix. These two logical database parts hold, in its entirety, the system's knowledge of the GERMS data model.

As for the aforementioned logical --> physical link, this is effected by two components of the logical database. For any logical relationship, the pointer matrix directs the system to the appropriate physical database directory. Likewise, the symbol table directs the system to the index (and therefore to the records) which relates to any given logical entity.

In reckoning with the issue of logical database structure and substance, we realized the need for a means through which the logical database contents can be communicated to the system. In response to this need we developed a GERMS data definition language. This language is used in the creation of a schema definition program -- a formalized model description which is written by the human user and interpreted by the database machine. In developing the data definition language we capitalized on the GERMS lingual form which was mentioned earlier in this review.

By the end of chapter 12, we had described what we call the "basic" information system. This system has the ability to respond to unplanned ad hoc information requests provided that (1) the key values of the geographic area(s) of interest are specified, or (2) all instances of a given area type are requested. Beyond this

useful yet limited capability, the ability to handle associative queries, automatic maintenance of semantic integrity, system aided database merging, and user "views" (subschemas) had not been considered. These very issues, which we call system "amenities" are the focus of chapter 13.

First, we considered how the ability to respond to associative queries, i.e. queries which are based on values of properties, could be given to our system. We learned that, in awarding this ability to the system, the existing physical database must be expanded, but it need not be restructured. In other words, the physical configuration of our basic system is unaffected.

Next, we developed the logic of a semantic integrity checking mechanism. The software components of this mechanism determine, based on the system's knowledge of logical data structure, various ways in which the database contents should be related. These relations are then verified by the system. It was noted that said process is expensive, but it need be performed only once for a given data set.

Following this discussion of semantic integrity we devised a database merging facility. The GERMS logical structure is used to direct the blending of two databases

which represent different properties of geographic areas. The merging facility enables the system to then "generate" new geopolitical statistical data which was not represented directly in either of the initial databases.

Finally, we considered how user views can be incorporated into the utilization of our information system. We discovered that no adjustments need be made to the structure of our logical database, to the structure of our physical database, or to the format of our queries. In fact, it is only the schema graph -- that document which provides the user with a "logical map" of the database -- is affected. By employing these user views we can often insulate the user from the need to deal with complexity which is peripheral to the application at hand.

Concluding Remarks

It is our belief that the conduction of research which fails to hold the ultimate purpose of improving some environment is of little use. Meaningful research must have an eventual pragmatic based intent. The goal of the research discussed herein is to induce some nontrivial improvement upon the existing geographic oriented database

environment. The primary thesis of this work is that said environment deserves a special purpose modeling formalism through which logical models of geographic based data sets can be developed. The principle focus of the work lies in the development of such a formalism.

The formalism which was developed supports direct conceptual modeling of the logical contents and structure of a geographic database; it captures the semantics of geopolitical statistical data. A data model produced through the use of this formalism is constructed without concern for the physical data structure (physical data independence). Each concept represented in the logical model need not be represented similarly in the respective physical configuration.

Before the value of the GERMS formalism can be assessed, we feel that the creation of a special purpose formalism, that is one which is dedicated to the modeling of a single specific application environment, should be justified. In making this justification we will analogize, as was done many times above, to the more appreciable area of computer programming.

The need for a special purpose modeling formalism. As is the case with programming language applications, the nature of database applications is quite diverse: ". . . just as it was eventually recognized in programming languages that there is no one 'best' programming language, so it is also now generally considered that there is no one 'best' data model. Different data models may be appropriate for different users, different tasks, and so on" [TSIC82, p. 173].

The majority of high level conceptual data modeling formalisms (DMFs) which are in existence today could be called "general purpose" modeling formalism. If the respective models have any specific realm of expertise it is, as evidenced by the examples of appendix A, the business managerial environment. In general though, they are designed so as to be useful in a wide variety of applications. Consequently, one of their primary attributes is flexibility. This flexibility is gained at the expense of precision and specificity, however.

When a model is developed within the constructs of a general purpose DMF it often happens that some of the characteristics that are particular to the application are represented either awkwardlly or not at all. As the characteristics of the application become more unique to

the situation, a general purpose database model becomes less applicable. The general purpose schemas do not provide a natural representation of relevant real world concepts; the tool does not fit the task.

Database researchers are becoming increasingly aware of the value of providing a "natural" representation of the applcation [WONG77]. The naturalness or realism of a database representation aids the database user as well as fortifying the understanding of semantic integrity enforcement -- if database structure reflects real world structure, the database is easier to construct and modify.

It would seem, then, that the scope of conceptual DMFs should be widened so that they might model a greater variety of applications in a natural way. There is a problem with this tack, however: there exists a significant tradeoff between a modeling formalism's power and naturalness, and its complexity [HAMM81].

As noted earlier, programming language designers struggle with the same dilemma. "The style that a language should best assume is far too dependent on the human and machine environment in which it is used. And that environment must also affect the level of sophistication of the features of a language. To say that a language has a generally accepted set of arithmetic

features is not to say that it is ideal for use in work totally oriented to numerical analysis. The less general the application of a language, the less adequate a general purpose language for those applications" [ELSO73, p. 243].

The response from programming language architects to this phenomenon has been the development of a number of "special purpose" programming languages which are aimed at meeting the special needs of unique programming applcations (e.g. DYNAMO, GPSS for simulation; LISP, SNOBOL for list and string processing).

Obviously, each individual programming environment does not require its own special purpose programming language. Instead, programming "environment types" which are large in number and unique in constructual demands deserve special purpose languages. We believe that the geopolitical data management environment is an environment "type" which is large in number and unique in constructural demands. Consequently, the environment deserves its own special purpose modeling formalism.

The uniqueness of the application at hand revolves, for the most part, around the nature of the primitive objects of the data model. Entities are geographic areas, relationships represent structured spacial overlap of geographic areas. General purpose DMFs have been designed

with different entities and relationships in mind (again refer to appendix A).

Specifically in these general purpose formalisms, entities are expected to be the "objects of a business enterprise" (e.g. employee, department, inventory item). Relationships are supposed to logically interrelate these object kinds (e.g. works for, produces, employs). It is apparent that the semantics which should be captured by a geographic based data model differ greatly from those of the general purpose applications.

As for the remaining primitive objects of a data model, properties and values, these too stand apart from the norm. A given property in our geopolitical application is held by every entity of the application. The semantic of this property is constant for every entity to which it applies. In general purpose applications this is not the case, so general purpose modeling formalisms tend to overemphasize the importance, and thus the complexity, of handling properties.

Values in our geopolitical apoplication are statistical objects unlike their general purpose counterparts. Furthermore, values of properties of different entities are related -- the nature of this relation being defined by the relationship which joins the

entities. This concept is inherent in the application and should therefore be inherent in the formalism used to model the application. Again, data model structure should match real world structure.

To summarize, the GERMS -- a data modeling formalism which supports direct conceptual modeling of geopolitical statistical data -- is better suited to capture the semantics of this special application than are general purpose formalisms. Real world concepts which would be represented either clumsily or not at all using other formalisms are represented easily and naturally using the GERMS. In other words, the GERMS is a tool which does fit the task.

Significance. The significance of the research presented herein should be measured in terms of the environment towards which the research is directed. Our discussion is thusly centered around the geographic oriented database environment. In reckoning with significance of our work we will consider the non-machine implemented GERMS version first (that is, the version described in part II). Then, we will consider the work described in part III.

In its "non-mechanized" form the GERMS provides two products:

1. It provides a means through which a graphical model of a geographic data set can be created

2. It provides a "language" for describing and discussing the logical structure of geographic based data

By providing these products the GERMS will help to ameliorate the complexities that surround the application environment in a number of ways. These betterments will be realized in the use as well as in the design stage of the geographic statistical data set.

First of all, a graphical representation of a semantic data model furnishes a syntax-free description of the data in much the same way as a flowchart provides a syntax free representation of the semantics of a program. The GERMS graph therefore serves as a valuable documentation tool for geographic based data sets. Such documentation indicates which data sets are applicable to which situations. The graph provides the data technicians and (end) users with an answer to the question: Will this data set provide me with the information that I need? A

related question which the logical data model helps to answer is: Given the physical structure of the data set, where can I expect logical anomalies (e.g. the non-containment problem of chapter 8) to arise?

True, answers to these questions can be found in the standard documentation which accompanies geographic data files. Such sources provide operational definitions of the geographic objects (entity types) and describe the physical structure of the files.

Documentation of this type does not, however, present a straightforward description of the logical interconnections that exist among the set of geographic objects represented in the file. This information is often buried within the patois of the individual entity type definitions. An understanding of geographic semantics is therefore required, but not readily available to the users of geographic data.

In the absence of a logical data model it is difficult to insure that data technicians and users hold a proper understanding of the "meaning" of the data with which they work. The mere development of the graphical specification (GERMS graph) requires that the database administrator have a clear understanding of the semantic properties of the data [SU79]. Thus, it forces the

precipitation of any semantic ambiguity which could otherwise reside unnoticed within the complex structure of the underlying data set.

Also, the development of the logical framework provides the data technicians with insight as to the maintainance of data integrity; the graph indicates how the data items should be related.

As for the data users, the document circumvents the possibility of the semantics being hidden in the application programs. The resulting clearer understanding of the meaning of the data allows for the formulation of more meaningful queries [SU79]. In other words, the data model provides the data users with insights as to what queries can be posed to the data set, and also how they should be posed.

As a geographic database development tool, a graphical representation of a semantic data model provides a basis for database design [HAMM81]. Through the use of GERMS structures the database designer can investigate the nature of the logical structure of geographic data while being freed from the need to consider physical structural issues. This results in the development of a database which is more likely to meet the needs of the user community.

With regards to the value of the "GERMS language," it provides the data technicians and users with an unambiguous communication medium through which the above issues can be described and discussed. Questions and statements about logical geographic data structure can be posed in a formalized way within this language. Each GERMS "phrase" matches a GERMS graphical concept which in turn relates to a well-defined geographic semantic construct.

In summary, then, what the "non-mechanized" GERMS formalism provides to the geographic data environment is similar to what flowcharting provides to the programming environment. It furnishes a means for investigating, describing, and discussing logical structural form while being freed from the need to consider physical structure and related syntax.

Just as the usefulness of a flowchart extends beyond the program design stage, so does that of the GERMS graph extend beyond the database design stage. The representation serves as a precise documentation and communication medium throughout the entire life cycle of the database [HAMM81]. This documentation and communication medium aids in the data administrative and maintenance tasks as well as promoting effective data utilization.

If the GERMS formalism is machine implemented (the part III version), there are even more benefits to be reaped. Queries are posed "in terms of" the semantically rich data model, so they tend to be more meaningful [SU79]. For the same reason, the queries are easier for the user to form. Likewise, system responses are formed in terms of the data model so the understanding by the naive user as to the meaning of the information is improved [HAMM81]. In short, then, the methodology increases the utility of the database and increases the effectiveness of the utilization of the database.

As for aiding in the database administrative and upkeep tasks, the system provides automatic maintenance of the semantic (not just syntactic) correctness of queries. In other words, the system holds the ability to reject queries which do not make sense in terms of the meaning of the set of objects with which the query deals.

Also, the system provides automatic maintenance of the semantic integrity of the database contents by verifying that stored data value makes sense according to data meaning. In addition, the methodology described above simplifies the merging of conceptually modeled databases -- a process which is not well understood [LUM78].

The system supports logical redundancy without requiring that physically redundant data be stored. Use of logical redundancy allows for the data model to more closely meet the logical needs of the user. Avoidance of physical redundancy reduces storage space requirements and simplifies a number of data management tasks.

Finally, the methodology is easily implemented in an existing database management system. In appendix C we showed how the GERMS system described above can be implemented, with only minor modification, in the SIR extended hierarchical DBMS. It should be apparent that an existing network or relational system could be used in a similar way. The importance of this facet is difficult to deny.

APPENDICES

"The properties and modes of action of higher levels are not explicable by the summation of the properties and modes of action of their components taken in isolation. If, however, we know the ensemble of the components and the relations existing between them, then the higher levels are derivable from the components."

Ludwig von Bertalanffy
"Chance or Law"

A P P E N D I X A

APPROACHES AND SCHOOLS OF MODELING

The purpose of this appendix is to provide an overview of the basic approaches to conceptual data modeling. Each section furnishes a description of a specific modeling formalism or modeling school.

The Entity-Relationship Model

The Entity-Relationship (E-R) Data Model [CHEN76; CHEN77] adopts the view that the real world can be described naturally in terms of entities; and relationships among entities. Despite the fact that the model has it foundations in set theory and relation theory, it is not considered as an extension of the relational model [CODD70; CODD72] specifically. Rather, it finds its place as a generalization of all first generation data modeling formalisms (DMFs) which is aimed at providing a semantically rich "unified view of data" [CHEN76].

Upon its introduction, the model was presented as a database design and documentation tool and, as such, did not interface directly with a database management system. No formal data description language or query language was provided. Instead, constraints, structures, and operations of the E-R model were specified in a general set notation.

An E-R database, designed in this way, was then mapped, by hand, into a first generation (usually relational) database schema. What was gained by this process was a semantic-based specification of the database contents as well as a direct (vis. stepwise normalization) approach to canonical schema design.

It was argued that stepwise object decomposition as defined by the schema normalization process is a bottom up approach to schema design. Arbitrary unnormalized relations (data structure) are transformed, as directed by the semantic implications of functional dependencies, into a third normal form (information structure) representaion.

The E-R model, on the other hand, provides a top down approach. Here, a high level conceptualization of the world is used as the starting point in developing an E-R (information structure) representation [CHEN76]. In addition, the resulting E-R schema often captures much

more of the real world semantics than does its third normal form counterpart [1].

The extreme popularity of the E-R model has resulted in the development of machine compatible E-R database systems. Such implementations center around an E-R specification language called CABLE (ChAin Based LanguagE) [SHOS78]. Although CABLE provides us with a well-defined syntax through which the E-R modeling concepts may be described, we choose a more generic medium of discourse: sets and relations.

In the E-R data model abstract objects representing the generic structures of entities and relationships (among entities) are called entity sets and relationship sets, respectively. Thus the E-R notion of a set corresponds to the ubiquitous "object type" concept. The intensional structure of the database is described completely in terms of these two primitives when the E-R modeling formalism is applied. A network-graphical depiction of this intensional structure is called an entity-relationship diagram (ERD) [CHEN76].

In an ERD each entity set and relationship set is represented by, respectively, a rectangular box and a diamond shaped box. Every box in the ERD carries a semantically meaningful label. Information as to the

participation of entity sets in relationship sets is provided by (generally) undirected arcs which connect relationship sets to their constituent entity set(s). If desired, added clarity may be given by an arc label which specifies the role played by the entity in the relationship.

When viewed at this abstract resolution, a network graph containing two distinct node types, the constraints imposed by the E-R model are limited:

1. Connective links may exist only between nodes of different types (entity node-relationship node)

2. Each connective link is labelled with the maximum cardinality permitted for the respective entity set in the relationship set (many mapping may be specified as a variable, e.g. "N")

In light of the limited nature of these constraints we observe the following characteristics of relationships in the ERD network:

1. A relationship may be n-ary. More than two entity sets may be connected to a single relationship set (see figure A.1) [2].

2. A relationship may be recursive. An entity set may be associated with itself via a relationship set (see figure A.2).

3. A pair of entity sets may be joined by more than one relationship set. In this situation each relationship set can capture a different semantic (see figure A.3).

At the extensional or token level (see chapter 4), we can consider entity instances as being members of entity sets. Entity set membership, which is determined by a predicate, need not be disjoint. Thus a token entity can be a member of more than one entity set.

If e(i) is an entity with membership in the entity set E(i)

$$(e(i) \varepsilon E(i))$$

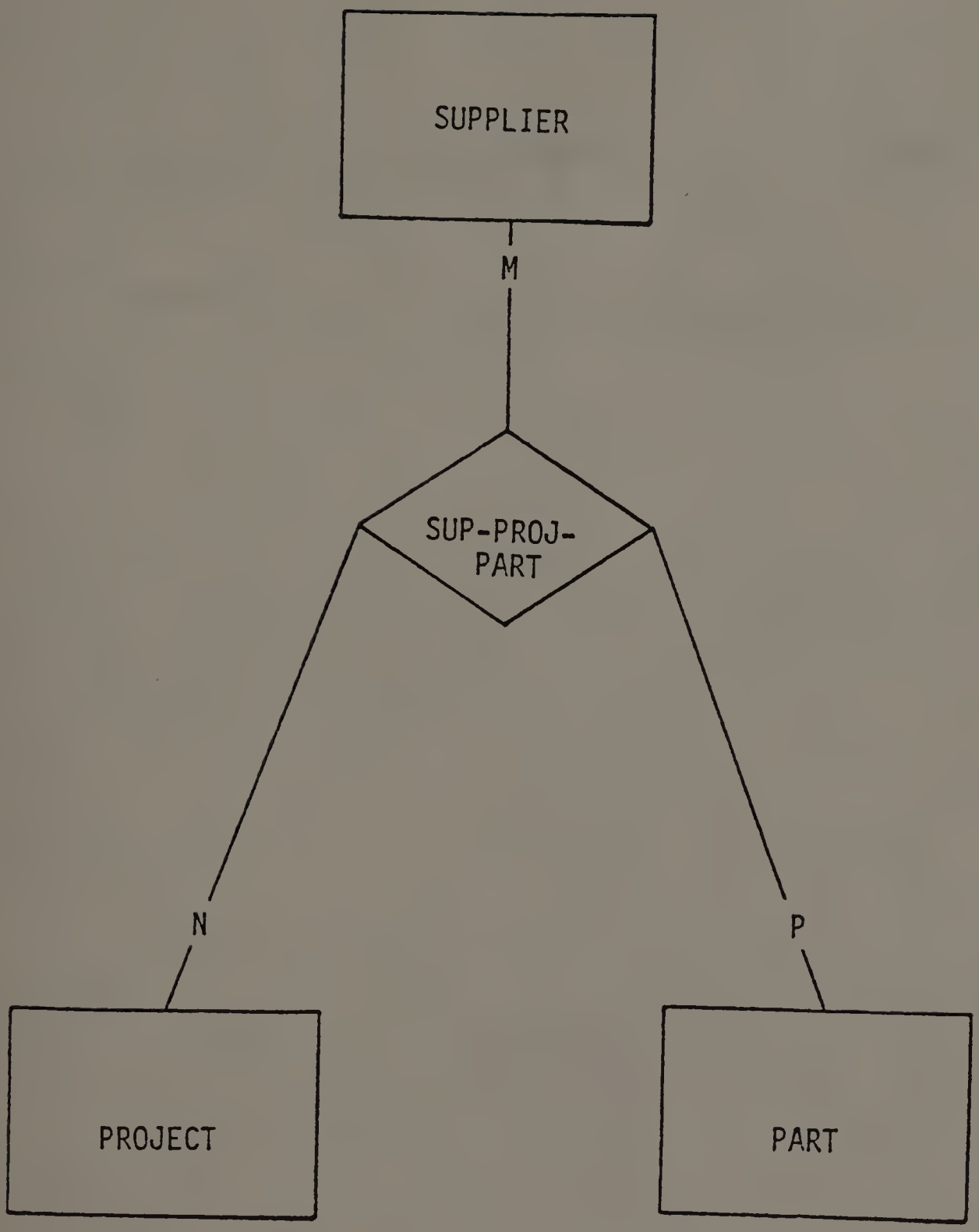


FIGURE A.1

A TERNARY RELATIONSHIP

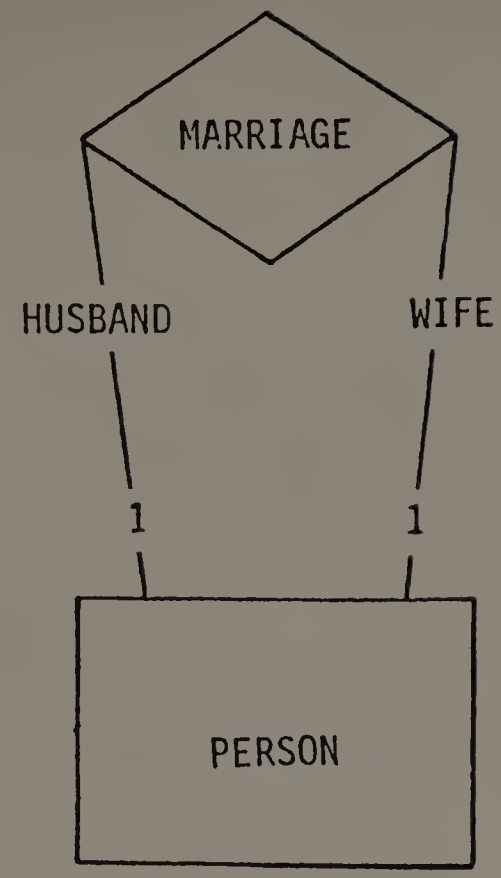


FIGURE A.2

A RECURSIVE RELATIONSHIP
(NOTE THE USE OF ROLES)

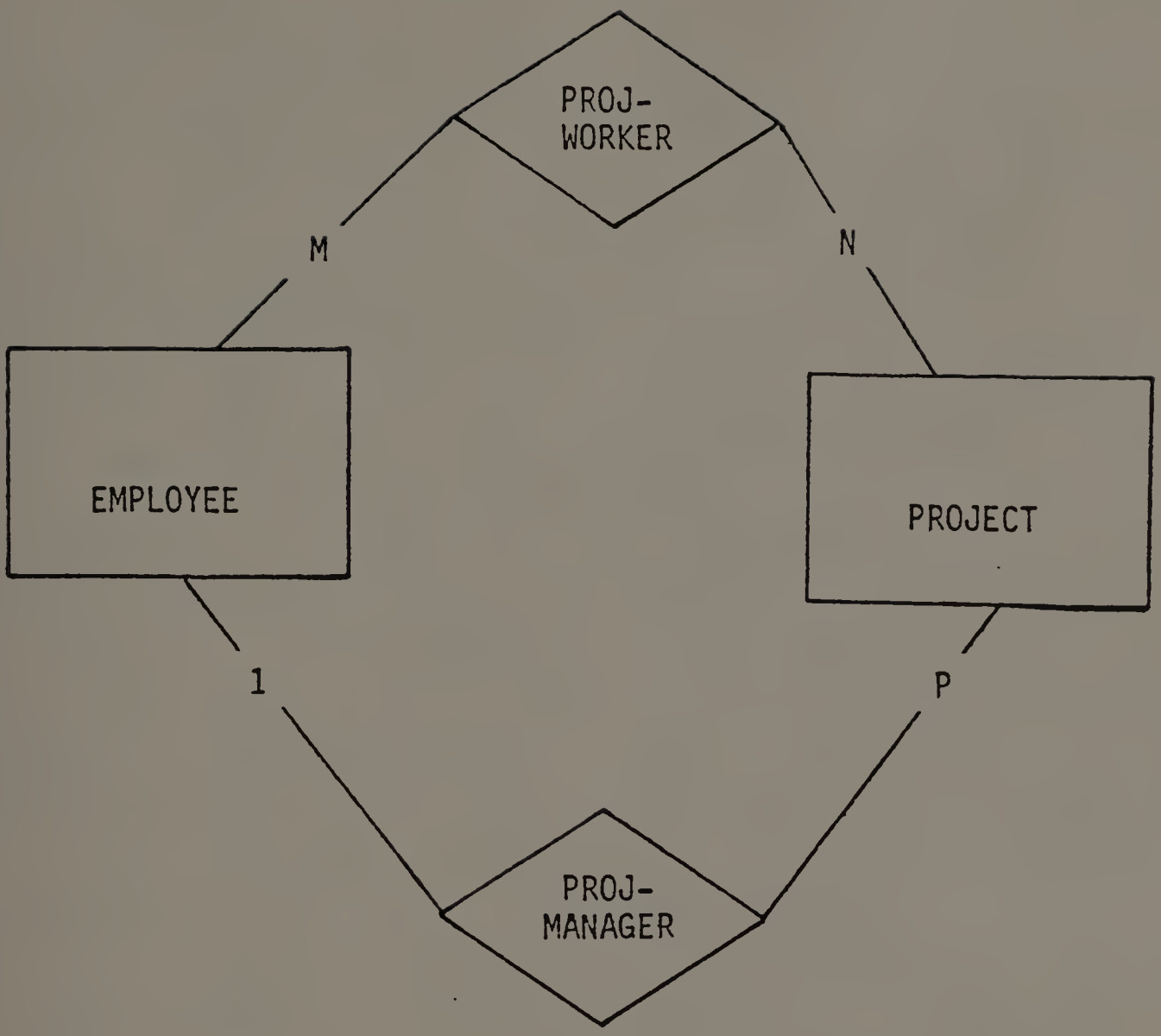


FIGURE A.3

TWO ENTITY SETS JOINED
BY TWO RELATIONSHIP SETS

then an n-ary relationship set (R) can be defined as a mathematical relation of degree n among the entity sets [TSIC82]. Thus:

$$R\{[e(1),e(2)...e(n) \mid e(1) \varepsilon E(1),e(2) \varepsilon E(2)...e(n) \varepsilon E(n)\}$$

Each tuple (called a relationship) of the relation R represents a particular instance of the relationship set. Our figure A.1 example could therefore be defined as a set of 3-tuples by means of:

$$\text{SUP-PROJ-PART} = \{[e(1),e(2),e(3)] \mid e(1) \varepsilon \text{PROJECT}, e(2) \varepsilon \text{SUPPLIER}, e(3) \varepsilon \text{PART}\}$$

When the relationship is recursive, as in the figure A.2 example, the E(i)'s need not be unique within R:

$$\text{MARRIAGE} = \{[e(1),e(2)] \mid e(1) \varepsilon \text{PERSON}, e(2) \varepsilon \text{PERSON}\}$$

A value set is a domain comprising a number of token values. The membership of values in a value set, which need not be disjoint, is determined by a predicate.

Value sets can be associated with both entity sets and relationship sets in the E-R model. The mapping between an entity set or a relationship set and the associated value set(s) is called an attribute [3]. Each attribute is assigned a semantically meaningful name which may be the same as that of the value set into which it maps.

Attribute mapping is functional in nature, so the result of the mapping is the identification of a single value token (or a single value tuple token). An attribute can define a mapping (f) from an entity set or a relationship set into either a value set or the cartesian product of a number of value sets [CHEN76]:

$$f: E(i) \text{ OR } R(i) \rightarrow V(i) \text{ OR } V(i,1) \times V(i,2) \times \ldots V(i,n)$$

As an example of the situation where an attribute maps into the cartesian product of value sets consider that the attribute ADDRESS maps into the cartesian product

of the value sets STREET-NUMBER and STREET-NAME (see figure A.4). Note that the cartesian product of value sets, as referenced here, holds a close correspondence to the early CODASYL notion of a "property space" (P) (see [CODA62]). Obviously, it is allowed for more than one attribute to map into the same value set(s).

As for the extensional representation of entity sets, each can be conceptualized as a two-dimensional table called an entity relation [4]. Each row of an entity relation (called an "entity tuple") represents the occurrence of a token of the entity set. The columns of the table correspond to the values of the underlying value sets as defined by the attribute mapping. When relationships themselves have attributes, columns in the "relationship relations" (elements of the "relationship tuples") have an analogous meaning.

The E-R model allows for the expression of two types of inter-object dependency: "existence dependency," and "identification dependency." In the former case, the existence of an instance of one entity set presupposes the existence (within the domain of discourse) of an instance of another entity set with which it is associated [5]. A common example is found in the existence dependence of a TAX-DEPENDENT entity upon an EMPLOYEE entity. When

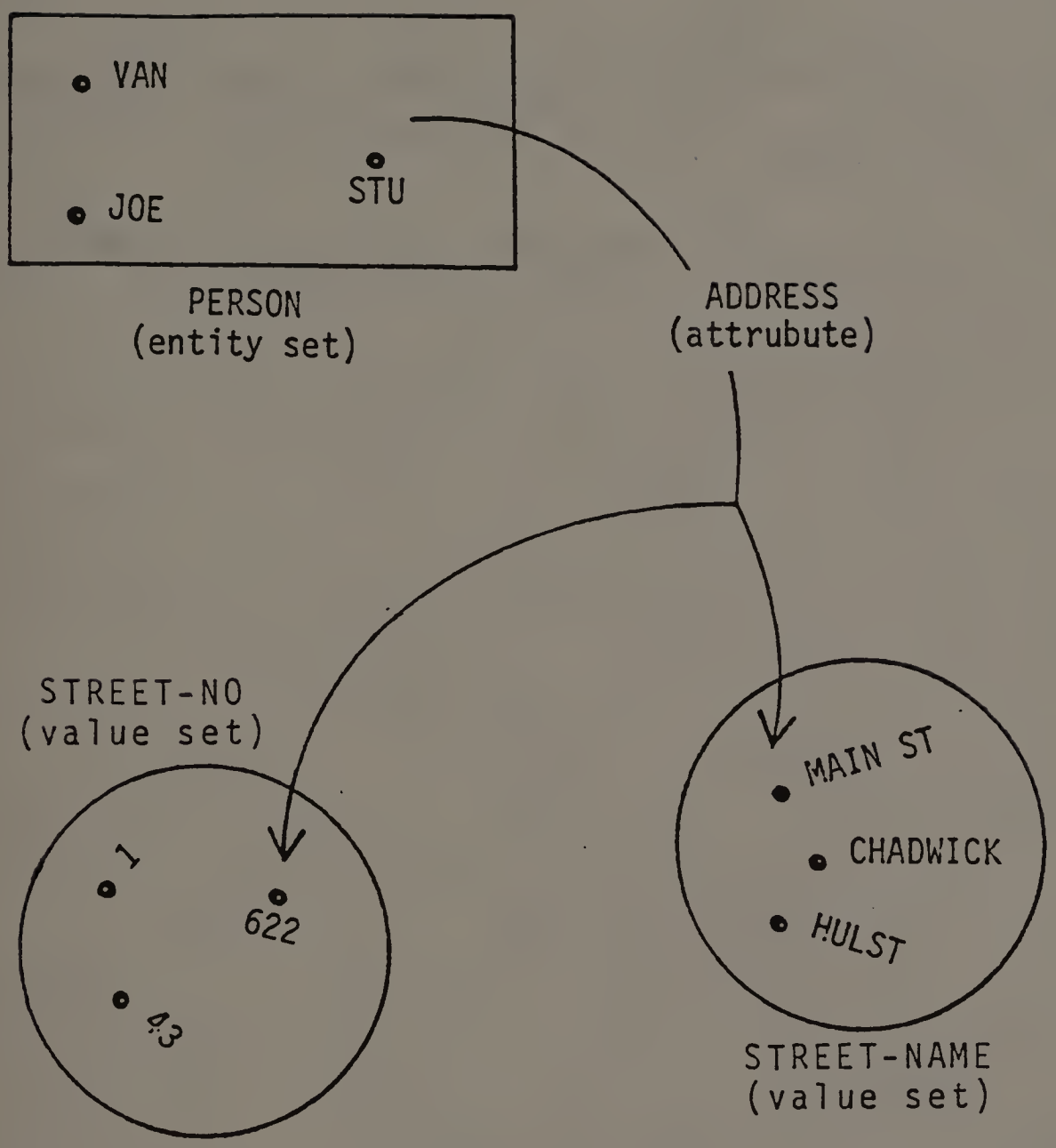


FIGURE A.4

FUNCTIONAL MAPPING OF AN ATTRIBUTE FROM AN ENTITY SET INTO TWO VALUE SETS

existence dependency is exhibited in the E-R model, both the involved relationship set and the depending entity set are said to form "weak" relations.

The second form of dependency, identification dependency, is used to model the situation where the instances of an entity set can not be identified by the values of its own attributes. Instead, it is identified by its association with instance(s) of another entity set. If there is an identification dependence between two entity sets, then there is also an existence dependence between the objects. The reverse implication does not hold, however.

Beyond object dependency constraints, semantic integrity can be enforced by specifying attribute constraints. Constraints of this type can take a number of different forms. One such constraint is implicit within the domain definition of the value set into which an attribute maps.

Value sets can be restricted as to primitive data type and range of values. For example, the domain of the value set GRADE-POINT-AVERAGE into which the attribute GPA of the entity set STUDENT maps could be restricted as:

TYPE: real numeric

RANGE: $0.0 \leq V(i) \leq 4.0$

Inter-object value constraints can also be expressed in the E-R model. Consider, for example, how we could express the semantic constraint that each manager must also be an employee:

$$\{Name(e) \mid e \in MANAGER\} \subseteq \{Name(e) \mid e \in EMPLOYEE\}$$

As for more complex specifications, we can define aggregate constraints to hold between entity sets that are qualified by membership in a relationship. For instance, the departmental payroll must equal the total of the salary values of the employees that work in that department:

$$Payroll(e(1)) = \Sigma\ Salary(e(2)) \mid e(1) \in DEPARTMENT,\ e(2) \in EMPLOYEE,$$
$$[e(1),e(2)] \in WORKS\text{-}IN$$

In conclusion, it should be noted that the wide acceptance and popularity of the E-R data model is due to its mixture of simplicity with rich semantic expressive

power. ERD specifications are general enough to express real world constructs, yet they are not too far removed from effective machine representation. Furthermore, ERDs serve as a valuable medium through which users and system designers can communicate [TSIC82].

The Basic Semantic Data Model

From the viewpoint of the Basic Semantic Data Model (BSDM) [SCHM75], the world can be represented in terms of object instances:

{obj(i)}

and relationship instances:

{ir (obj(1),obj(2),obj(3)...obj(n))}

The token objects and token relationships which have common properties are grouped into object types and relationship types, respectively.

Given this introductory description, we are tempted to correlate the BSDM with the Entity-Relationship (E-R) approach. We will soon learn that the methodologies are quite different, however. The BSBM is very much more concerned with object characterization than is the E-R model which, incidently, is its chronologic successor.

Relationships, as they exist in the BSDM, have a direct semantic meaning. They can not be further decomposed. Relationships represent fundamental facts about the involved objects.

A relationship instance, ir(obj(r), obj(d)), is a "depending relationship" if the existence of one constituent object, say obj(r), implies the existence of (1) the relationship instance itself, ir(obj(r), obj(d)); and (2) the remaining object, obj(d). In this case, obj(r) is called the "ruling part" and obj(d) is called the "depending part" of the depending relationship instance.

There are two distinct kinds of relationship instance: characteristic relationships and association relationships. This relationship partioning is mutually exclusive and exhaustive.

Each instance of a chracteristic relationship is a depending relationship, say ir(obj(r), obj(d)), which holds the following qualifications:

1. The token obj(r) is the ruling part of the relationship

2. The remaining depending part token obj(d) participates either as the ruling part or not at all in other relationship instances

Here, the depending object is called a characteristic object.

An object which is not a characteristic object is called an independent object. An independent object is never the depending part of a depending relationship instance. Note that, given the above specification, both independent objects and characteristic objects can serve as the ruling part of characteristic relationships. In other words, characteristic objects can themselves be characterized.

Both independent objects and characteristic objects can be either simple or complex. A simple object is one which has no characteristics (does not act as ruling part in any characteristic relaionship).

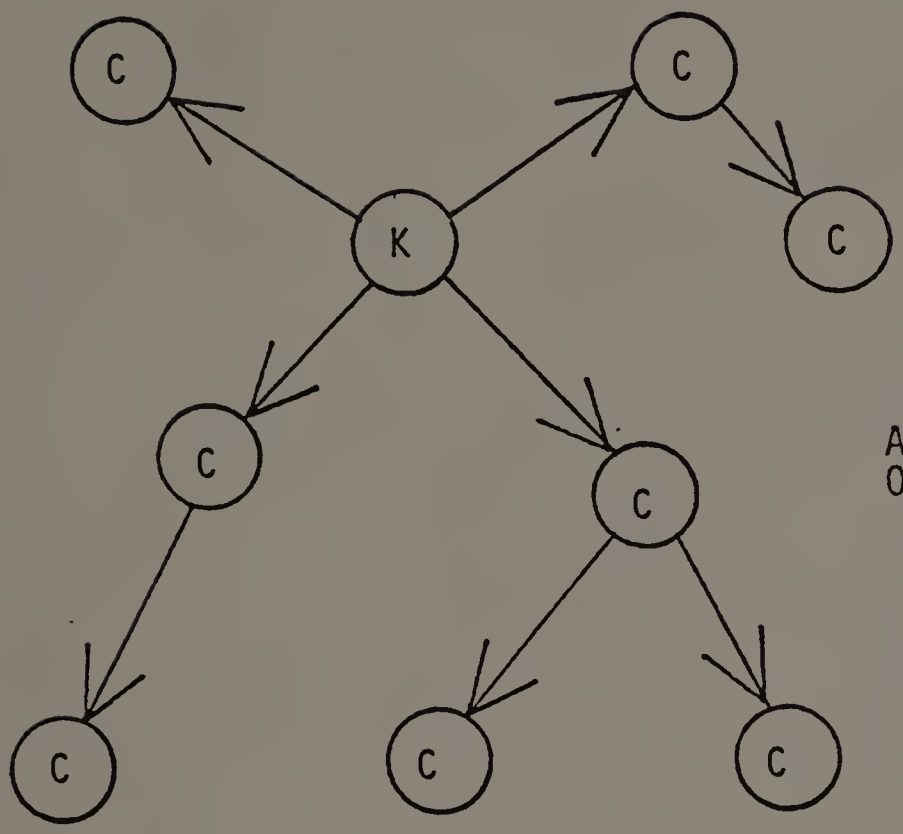
The definition of a complex object type includes, recursively, its characteristics extending to the simple object level. Thus, a complex independent object type comprises a kernel independent object type; all complex and simple characteristic objects with which the independent object participates in characteristic relationships; all complex and simple characteristics through which these complex characteristics are characterized; and so on.

A complex characteristic object is defined in a similar recursive manner [6]. A simple characteristic object comprises itself. The notion of a simple independent object type, a class of noncharacteristic objects which have no characteristics, makes little sense and is not considered (see figure A.5).

Kernels of complex independent object types can be interconnected only by means of association relationships. Corresponding association instances can be created only if the respective token objects exist [7]. There are no formal restrictions on the deletion of association instances, however.

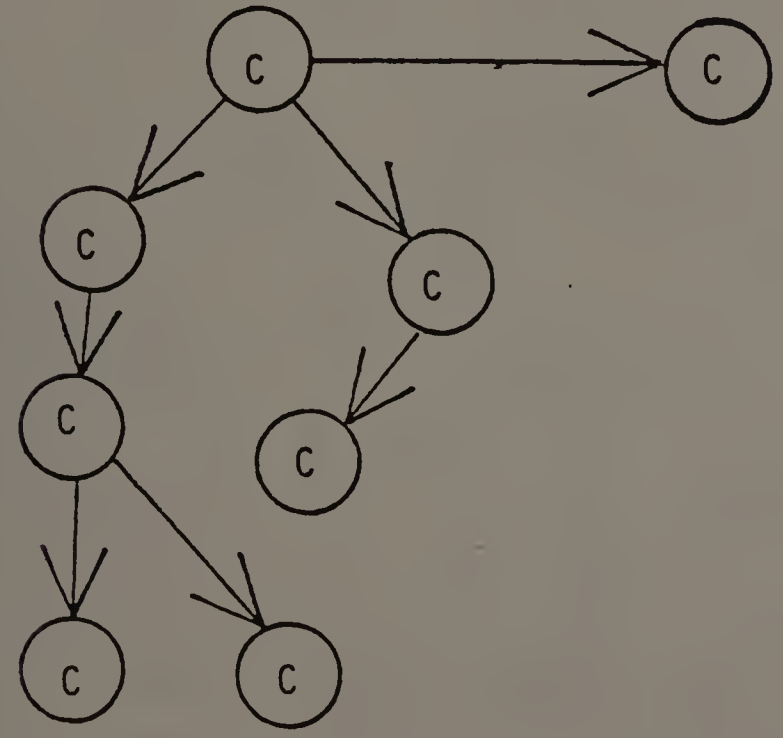
The objects of the BSDM have a natural (semantic) kinship to the standard database update primitives (insert, delete, modify). Independent objects represent

THE OBJECTS OF THE BSDM



K = KERNEL
C = CHARACTERISTIC

AN INDEPENDENT OBJECT



A COMPLEX CHARACTERISTIC OBJECT

A SIMPLE CHARACTERISTIC OBJECT

FIGURE A.5

the entities of the real world. As such, they are the exclusive focus of *insert* and *delete* directives. *Modify* operations are not applied directly to independent objects; they refer to *characteristic objects.*

This concept is elucidated when we consider that there exists a "semantic dependency" of each characteristic object upon its independent object. An independent object instance "determines" the characteristic objects which describe it -- an employee instance determines her or his salary, address, and hair color. Consequently, existence of a characteristic object depends (if semantic integrity is to be preserved) on the *a priori* existence of the independent object which it characterizes.

As their name implies, complex independent objects are unaffected by operations performed on other independent objects. They exist orthogonally in the model. The existence of an independent object does depend on the existence of at least one association instance in which it participates, however. In other words, "free floating" independent object tokens are not allowed.

The formalism that is based on these primitive notions includes the specification of a variety of special types of association and characteristic relationship.

Many of these types have their foundations in artificial intelligence (AI) and therefore have well-defined semantic connotations.

An association instance can be typed as "subconcept of" or "part of," for example. Specification of a subconcept of association implies that the characteristics of the superconcept independent object are inherited by the subconcept object. Specification of a part of association also implies the potential for the derivation of more detailed properties of constituent objects.

Application of the BSDM formalism to a database allows for the database to be visualized in different levels of abstraction. In the most abstract or coarsest level, the data comprises only independent objects and accompanying associations. At this level, complex and simple characteristics are ignored completely. Note that ignorance of association relationships is not allowed.

At the next lower level of abstraction, the direct (independent object is ruling part of depending relationship) characteristic objects are included. Here, the complex nature of complex characteristics is not considered. Thus, characteristic objects are not themselves characterized.

As the conceptualization of the database becomes less and less abstract, characteristics of complex characteristic objects continue to come into consideration. Eventually, at the finest (least abstract) level, the database is considered in full detail.

The process described above is somewhat similar to stepwise movement down the aggregation hierarchy (aggregate decomposition) as described in [SMIT77a] and [SMIT77b] (see chapter 5). It is an extremely restrictive case of aggregate decomposition, however, because the BSDM demands that only the topmost aggregate hierarchical level be allowed to include independently existing real world objects.

The BSDM is a formalism which can be used to provide a semantic description of relational database structures and operations. In the absence of such a description, the basic relational model [CODD70; CODD71; BEER78] is purely mathematical.

Functional dependency of relational attributes corresponds directly to semantic dependency of BSDM objects. Relational schema normalization achieves the separation of complex independent objects from one another and from associations.

After conversion to normal form the representation of each BSDM independent object token and association instance occurs only once in the relational database. With this in mind, we achieve an understanding of how the mathematically based process of normalization eliminates the occurrence of semantic based insert and delete anomalies in a relational environment.

The Semantic Association Model

The Semantic Association Model (SAM) [SU79] emulates the enterprise's real world in terms of a set of interrelated "concepts." A concept is a physical or abstract thing or event. Concepts are of two basic types: atomic and non-atomic. An atomic concept is non-decomposable and its meaning is both understood by and of consequence to the enterprise. A non-atomic concept is one whose meaning is described or defined in terms of other atomic or non-atomic concepts.

The grouping of concepts to define a non-atomic concept is called an association. Thus, each occurrence of an association relates to the formation of an occurrence of a non-atomic concept. Associations are of a

number of different types. In fact, the entire SAM is based on distinguishing types of associations and the concepts that they form. The wide variety of association types allows for an extremely detailed description of the semantics of data abstraction (see chapter 5).

A set of similar concepts is grouped by means of a "membership association" to form a concept class (CC). Each concept class is represented by a CC node in the SAM schema. The data definition of a CC is tripartite. It includes the specification of:

1. A CC name

2. A data type primitive

3. A domain definition

Note that the latter two items represent semantic integrity constraints.

A SAM database management system (DBMS) utilizes these contraints in determining the validity of the use of relational operators (e.g. EQ, LT, GT, NE) within database transactions. Such object comparison is restricted to be within a CC or between CCs that belong to a higher level CC. The following example should help to

vivify this point.

Consider the CCs "SUPPLIER-NAME," "CUSTOMER-NAME," "NAME," and "CUSTOMER-ADDRESS." Objects within any one of these CCs are comparable because they hold identical type and domain descriptions. Inter-CC comparison is not as simple; we must utilize the integrity constraints. SUPPLIER-NAME and CUSTOMER-NAME are comparable because they belong to the higher level NAME CC. SUPPLIER-NAME and CUSTOMER-ADDRESS objects are not comparable, on the other hand, because they do not share a common CC membership (see figure A.6).

The grouping of heterogeneous CCs, all of which serve as attributes in the definition of an "entity type," is called a characterization association. From this description we realize that a SAM entity type is a group of token entities which are characterized by a common set of attributes. This definition of entity type does not conflict with common usage.

The attributes in a characterization association can be atomic or non-atomic concepts. For example, the entity type EMPLOYEE is described in a characterization association by the concept classes EMP-NO, EMP-NAME, DEPARTMENT, and INCOME. The CC EMP-NAME is non-atomic in that it comprises the CCs FIRST-NAME and LAST-NAME. The

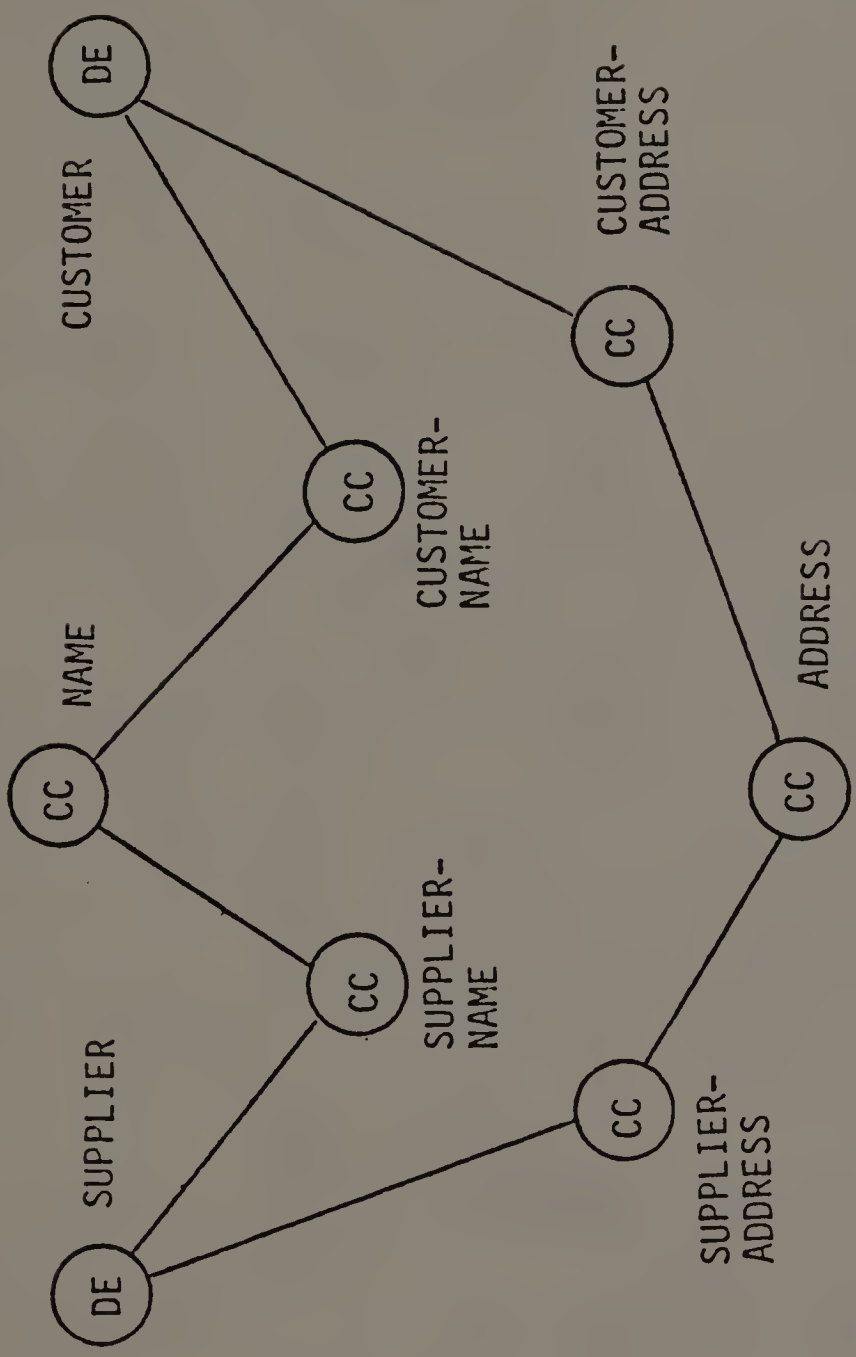


FIGURE A.6
MEMBERSHIP ASSOCIATIONS

characterization association most closely resembles traditional aggregation abstraction (refer again to chapter 5).

A characterizing entity (CE) is an entity which exists merely to characterize another entity. It therefore has an existence dependence on the entity it defines. A defined entity (DE), on the other hand, holds independent status (see figure A.7).

In terms of CE semantics, the DBMS enforces semantic integrity by:

1. Rejecting the entrance of a new CE token object into the database unless the characterized token object exists

2. Deleting all CE tokens when the respective characterized entity token is deleted

3. Limiting access to the CE object by requiring that it be accessed (referenced) "through" the characterized entity

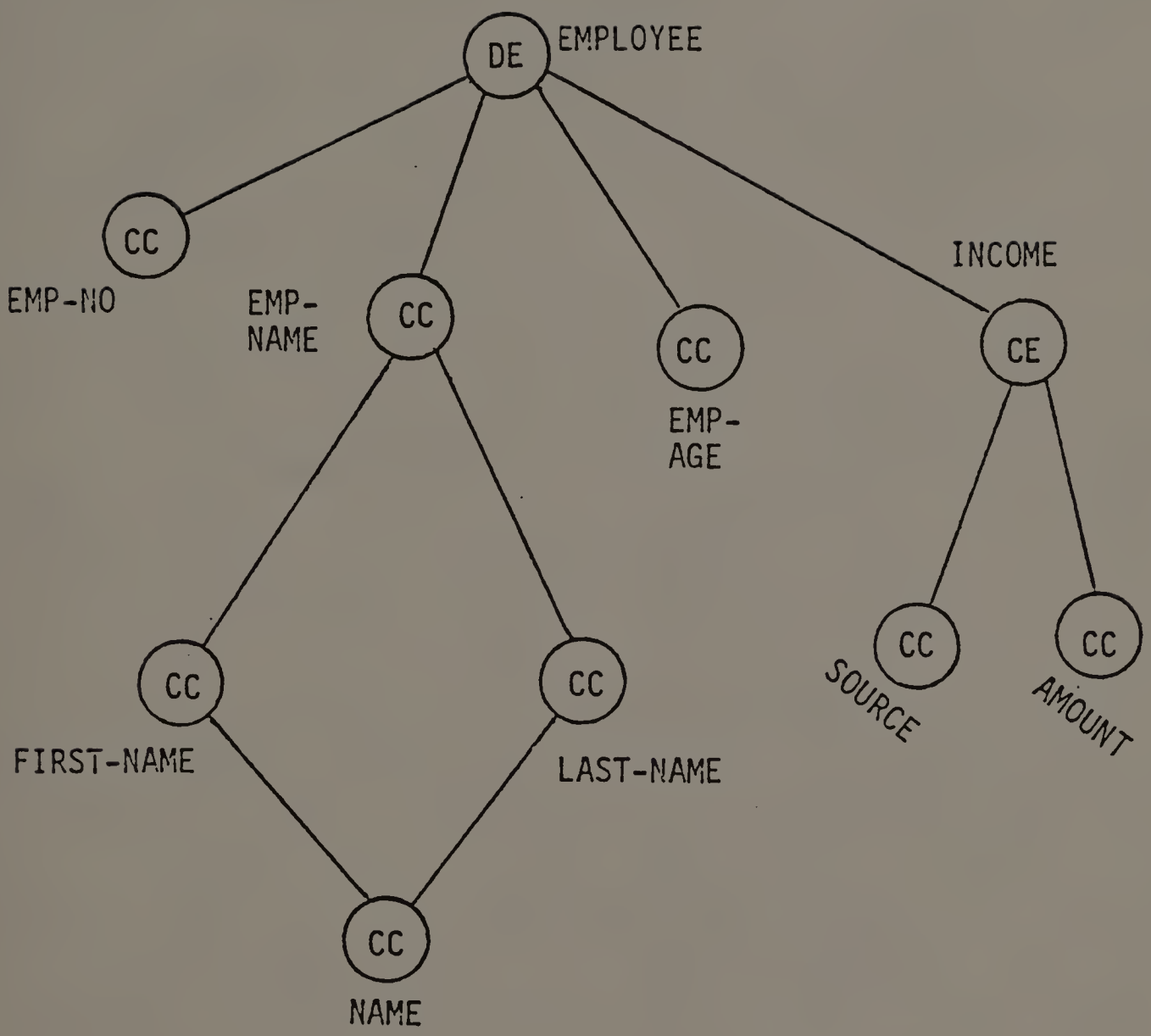


FIGURE A.7

A CHARACTERIZATION ASSOCIATION

The grouping of similar or differing entity types which represents the interaction of the types such that each type performs a specific function in the interaction is an "interaction association." The concept described by the interaction is an entity interaction (EI). The definition of an EI may include specification of the mathematical mapping among involved entities.

The types of function performed by the entities particpating in an EI are similar to those used in artificial intelligence semantic net applications. Some examples are agent (AG), direct object (DO), affected object (AO), and modifier (MD). Schema links between the EI node and participating entity nodes can carry semantic tags which specify the function of the participating entity in the interaction (see figure A.8).

With regard to EI semantics, the DBMS can enforce semantic integrity by:

1. Rejecting entrance of a token EI object unless participating entity tokens exist in proper cardinalities

2. Allowing the interaction data to exist beyond the deletion of the functional objects

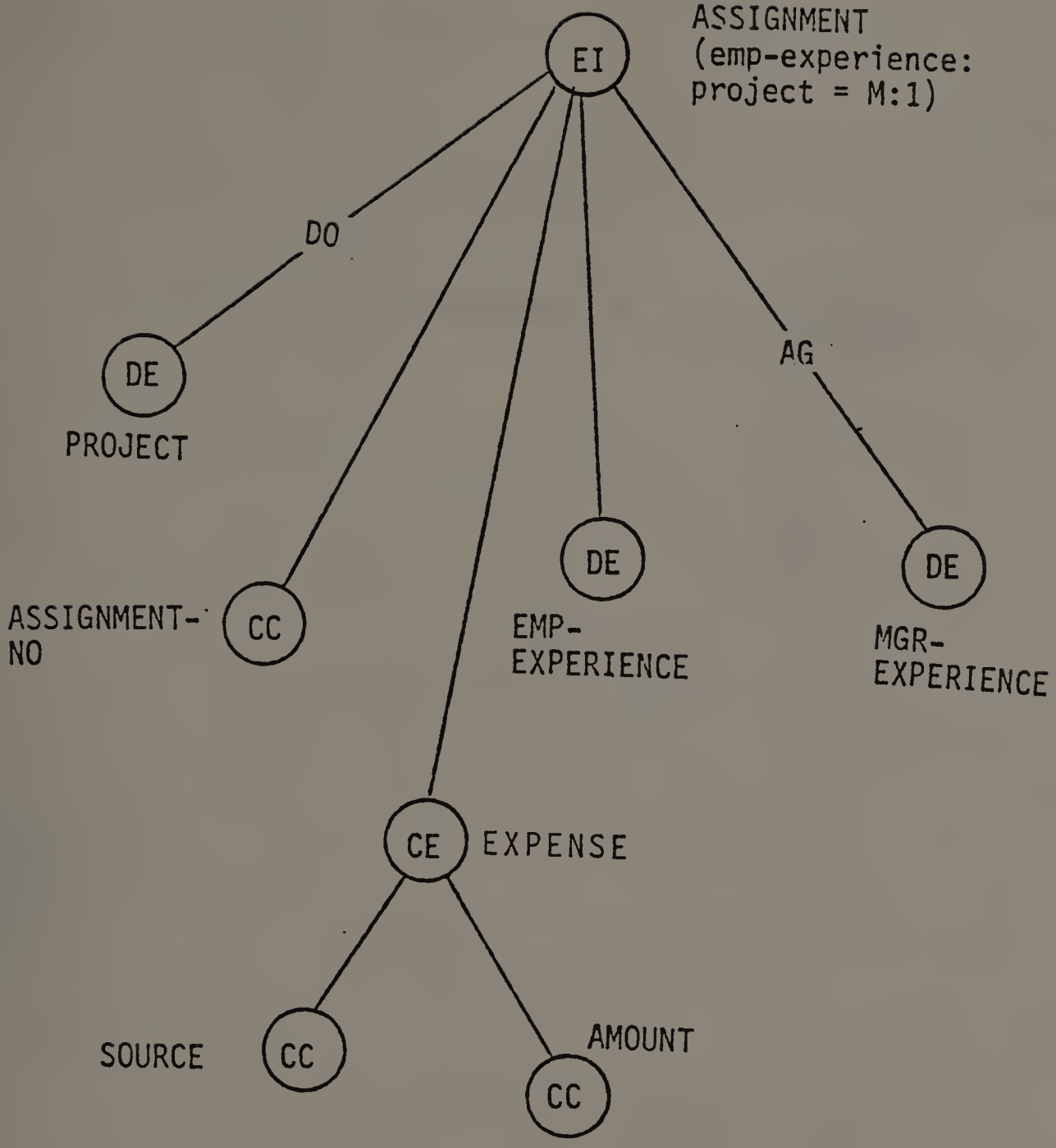


FIGURE A.8

AN INTERACTION ASSOCIATION

The significance of item 2 is not immediately obvious. If a separate EI object were not defined, information about the interaction would be represented within the definition of one of the functional entities. For instance, information about the sale of an item would be represented as information about the agent of the sale, say within an EMPLOYEE token. In this case, if the employee token is deleted, we lose information about the interaction (sale). By treating the interaction as a separate object which can exist beyond the existence of its creators, we do not suffer this consequence; the sale data is retained.

The set theoretic grouping of entity types which represents a common set of real world objects is called a set relationship association. The concept that is described by such an association is called a "set relationship" (SR).

Knowledge as to the meaning of the association is provided by semantic tags on the links which connect the SR concept to the participating concepts. These tags define the relationship between a constituent object and the SR object as being of type ST (set), SB (subset), SI (set intersection), SE (set equality), or SX (set exclusion). Figure A.9 provides an illustration.

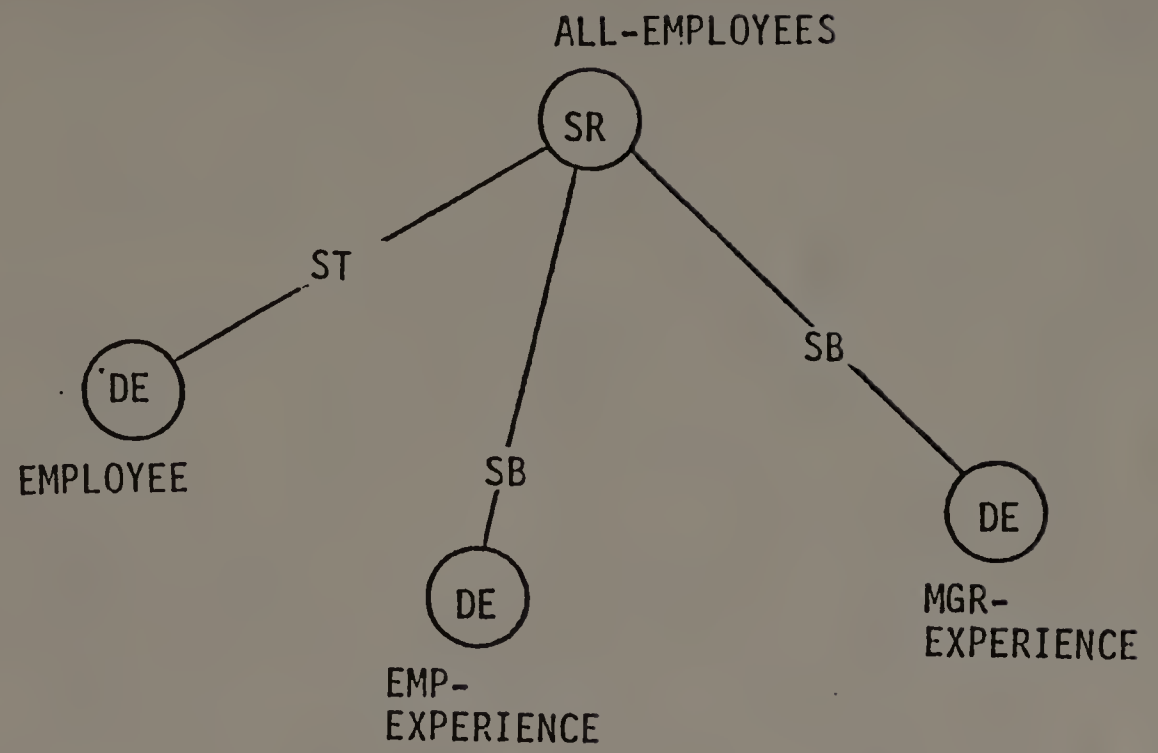


FIGURE A.9

A SET RELATIONSHIP ASSOCIATION

The implications of these tags upon the semantic integrity constraints that can be enforced by the DBMS are obvious. An SE tag, for example, defines that each token which is a member of one linked entity type must be a member of both linked entity types. An SX tag, on the other hand, states that an instance may not be a member of both types. An SI tag defines that a token object can belong to one type or to the other type or to both types. An SB link defines an existence dependence between the related entity types.

Concepts which serve as components of another concept are grouped together to form a "composition association." The concept whose assembly is represented by such an association is called a "composition" (CP). Each CP has one instance only.

A composition association differs from a characterization association in that the constituent objects of the former type are allowed independent status. This is an important semantic distinction. The composition association models how objects are "made up of" other objects which exist in and of themselves.

In a manufacturing environment, for example we can think of an assembly as being composed of various subassemblies; each of which may be composed of

subassemblies; and so on until we reach some level of primitive part. Each level of composition deserves representation and independent existence in its own right (see figure A.10).

Phrased another way, the composition of subassemblies and primitive parts to form an assembly represents the fact that the parts have been combined. It does not represent the fact that the parts have come into relevant (to the enterprise) existence. The aforementioned characterization association serves to model the latter situation. A consequence of the semantics of the composition association is that the DBMS allows for the independent insertion, deletion, and modification of component objects. Sibling components of the CP are unaffected by such changes.

The semantics of collective or aggregate attributes can be described by a CP (composition) which has only one component entity type. Here, the single CP represents the compilation of token component entities in the formation of a type. This collective representation can be characterized by means of characterization associations. The respective CC objects depict concepts which would be traditionally represented by DBMS "aggregate functions" (e.g. count, mean, sum). Figure A.11 provides an example of this.

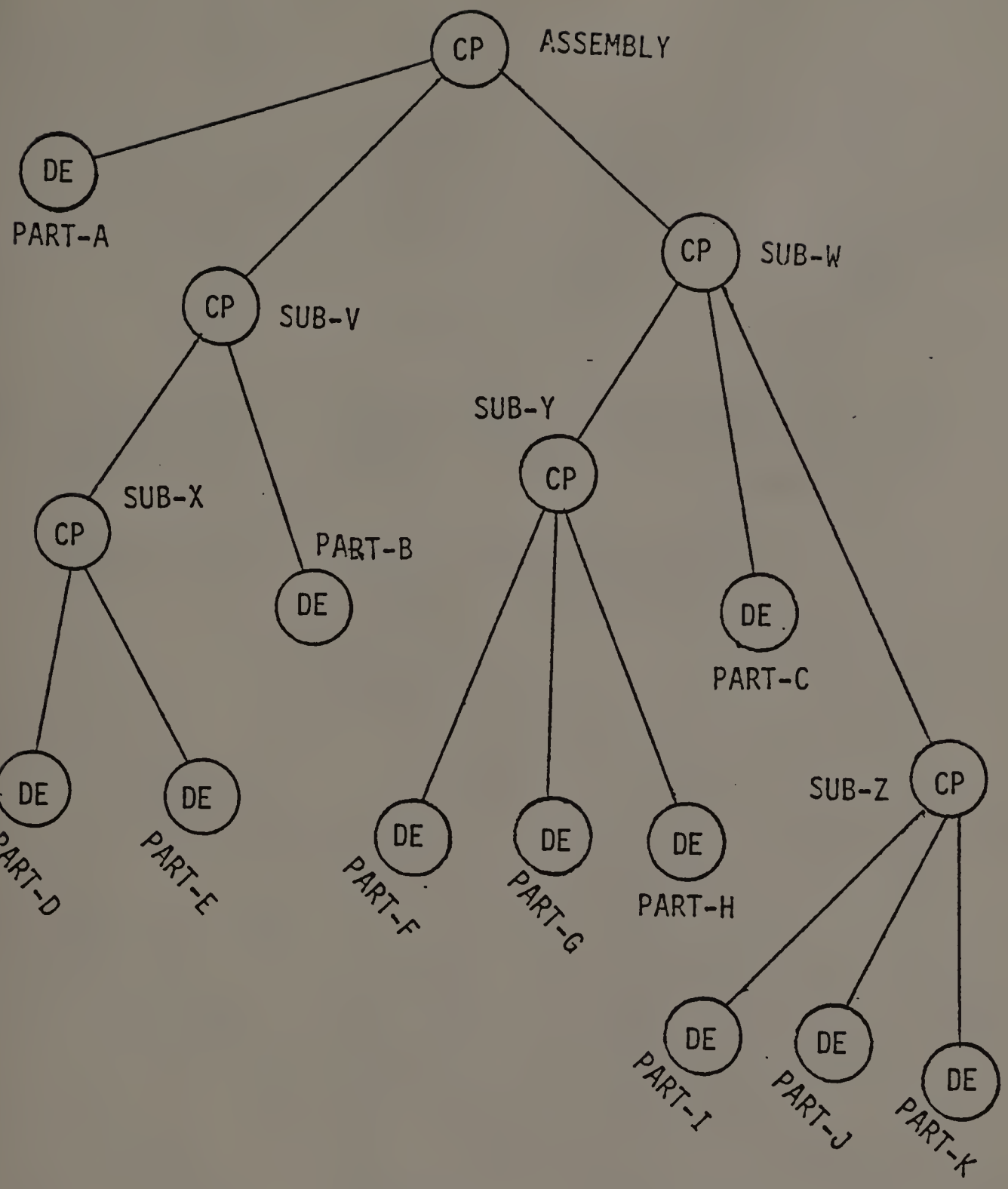


FIGURE A.10

A COMPOSITION ASSOCIATION

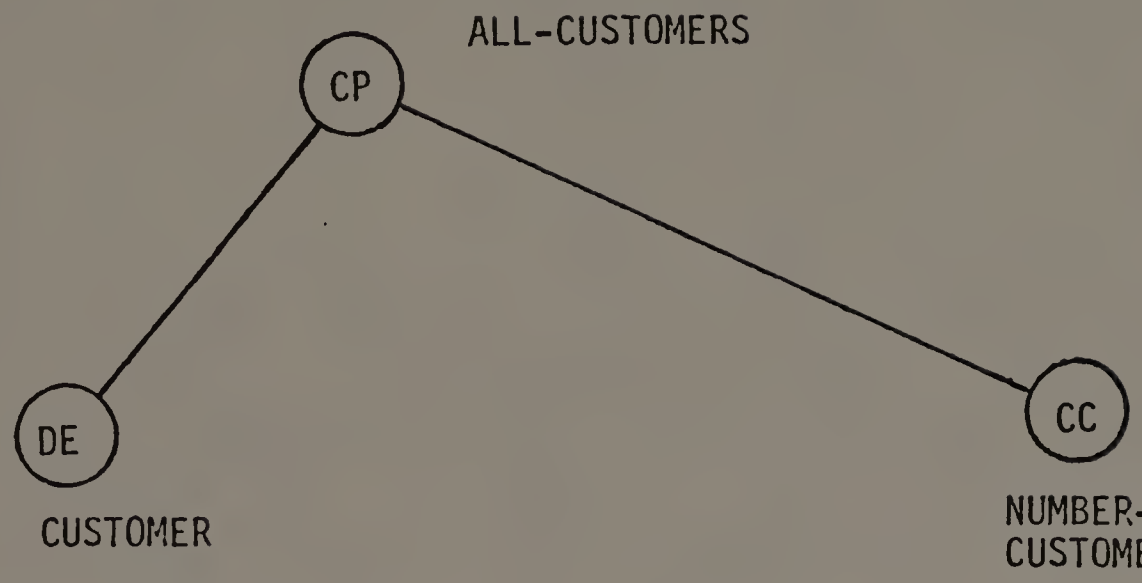


FIGURE A.11

USING A COMPOSITION ASSOCIATION TO
DEFINE AN AGGREGATE ATTRIBUTE

When one concept can be identified as being the cause of another (effect) concept, the semantics are captured by a "cause-effect association." Such an association defines the occurrence of a concept called a causation (CF). The links which connect the participating cause and effect concepts are labelled with the semantic tags "CA" and "EF," respectively (see figure A.12).

The DBMS can use the cause-effect association to drive an update triggering mechanism. When the prerequisites of causation are satisfied by proper updating of a CA linked object, the automatic updating of the respective EF concept is triggered. The causation concept is somewhat similar to the notion of "event precedence" as used in the RM/T data model [CODD79] [8].

When concept(s) are the means by which an action concept is performed, the semantics are represented by an "action-means association." The concept that is described by such an association is called an action-means (AM). One constituent object is identified as being an action, while the other constituent object(s) are the means of performing the action. The connective links of the association are tagged as AC (action) or MAC (means of action).

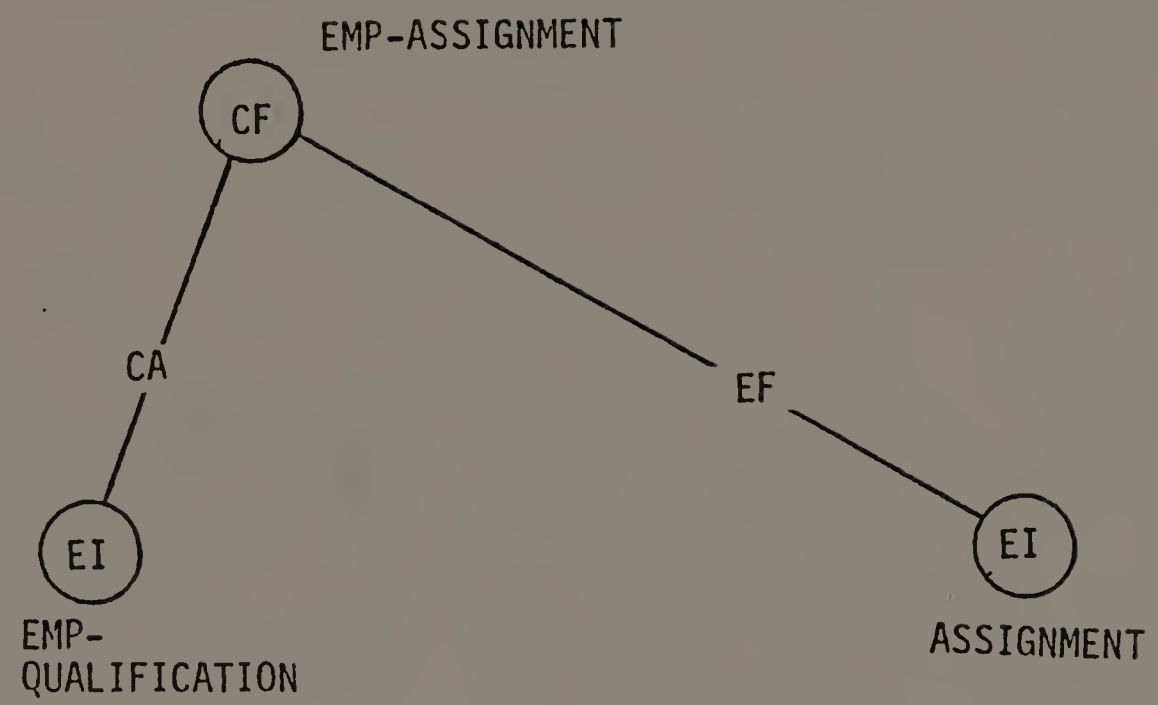


FIGURE A.12

A CAUSE-EFFECT ASSOCIATION

As an example, consider the concepts "PERSON-HOURS" and "PRODUCTION." If a given production token can be increased by prolonging one or more PERSON-HOUR tokens, then PRODUCTION serves as the action and PERSON-HOURS serves as the means of an action-means association. The AM concept "PRODUCTION-FACTOR" is defined by this association (see figure A.13).

A similar semantic association that can be represented is one in which one concept is the purpose for which an action-concept occurs. Such an association is called an "action-purpose association." The concept defined by an action-purpose association is called an action-purpose (AP).

Consider the concepts "SALES-EFFORT" and "SALES-QUOTA." Sales effort can be thought of as being an action which is performed in an attempt to reach a sales quota (purpose). These concepts, grouped in an action-purpose association, define an AP concept, say "SALES." Figure A.14 depicts this association.

The integrity constraints and triggers that should accompany the action-means and action-purpose associations are situation specific. As such, they are not specified as a part of the formalism. Since the association types are specifically recognized by the system, however,

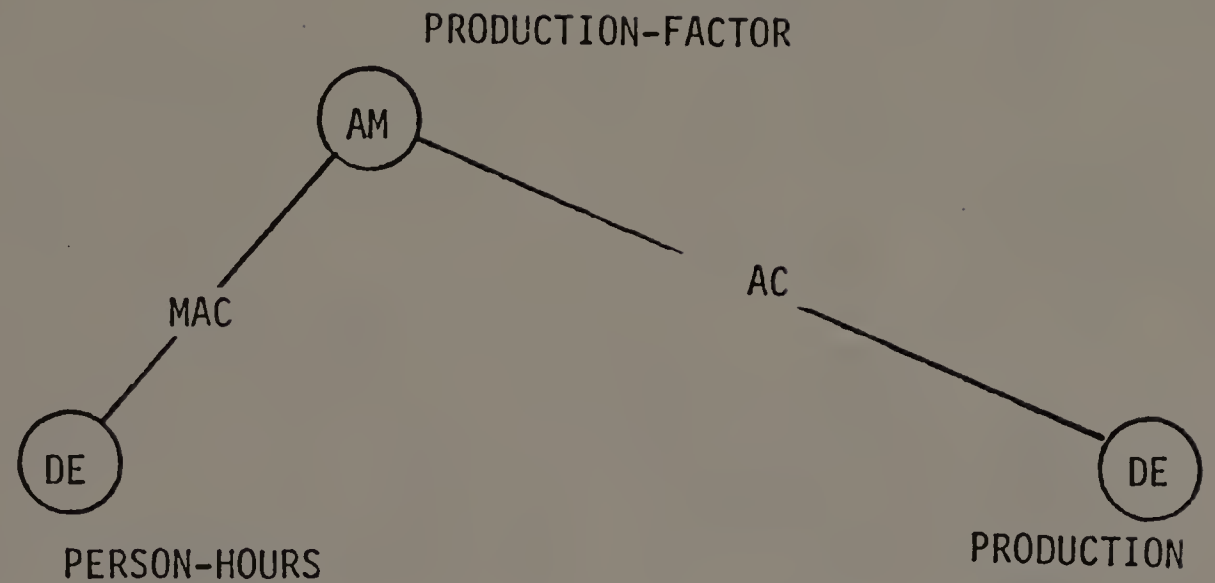


FIGURE A.13

AN ACTION-MEANS ASSOCIATION

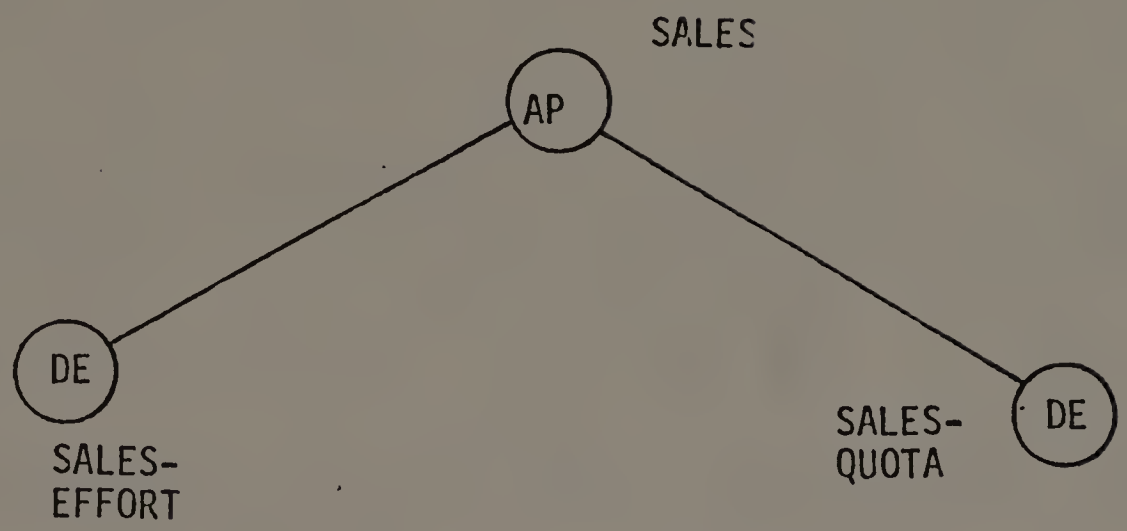


FIGURE A.14

AN ACTION-PURPOSE ASSOCIATION

appropriate semantic integrity triggers can be proceduralized easily at system implementaion time.

The final and perhaps most semantically powerful association of the SAM is the "logical relation of implication association." The object that is defined by this association type is called a logical relation of implication (LRI). This association is used to represent the semantics of the situation where the existence of one concept implies (logically) the existence of another concept. Situations of this type can be phrased in an if-then statement such as "if A then B."

The links that connect constituent objects to the LRI in a logical relation of implication association carry the appropriate semantic tags. Conditional and consequential concept links are labelled "IF" and "THEN," respectively (an example is provided in figure A.15). When implemented properly the logical relation of implication association can provide the DBMS with a strong deductive ability.

For the most part, the semantic richness of a SAM schema is provided by inclusion of the action-means, action-purpose, and logical relation of implication constructs. Indeed, these constructs allow for the system to handle "how and why" queries vis-a-vis the simple

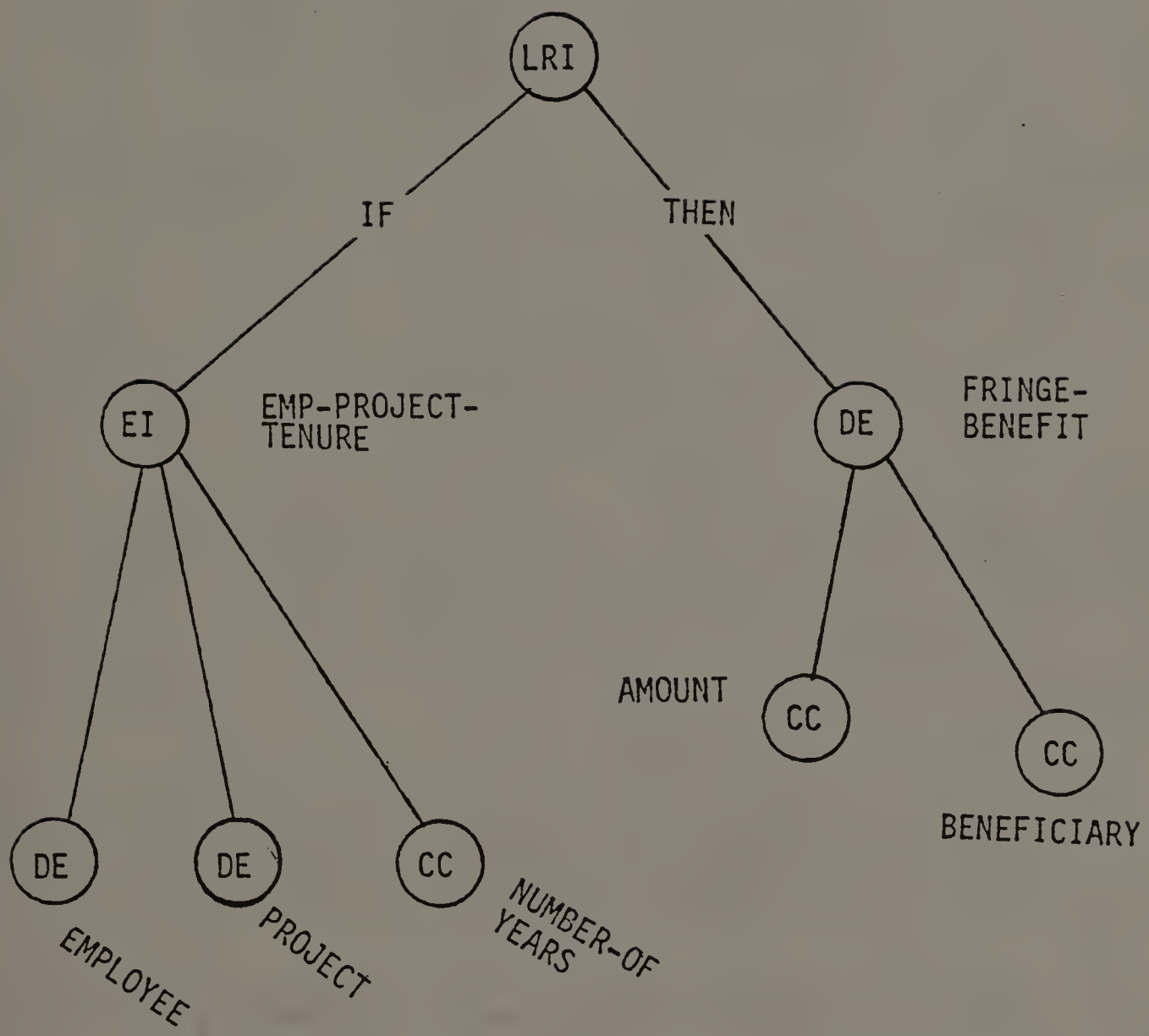


FIGURE A.15

A LOGICAL RELATION OF IMPLICATION ASSOCIATION

"what" queries that are answerable by traditional database abstraction structures [SU79].

The Semantic Database Model

The Semantic Database Model (SDM) [HAMM81] is founded on the notion that the real world can be modeled by describing the grouping of objects into meaningful classes. The fundamental building blocks of an SDM representation are as follows:

1. Entities -- reflect real world objects in the usual way

2. Classes -- are meaningful named collections of entities

3. Interclass connections -- structurally relate classes

4. Attributes -- describe the characteristics of, and relate, entities and classes

5. Derivation primitives -- provide for a formal definition of derived attributes and interclass connections

The SDM does not explicitly distinguish between aggregation and generalization abstraction. Rather, all abstraction is performed in terms of defining and associating classes of objects in various ways. At the most general level of abstraction we find the notion of base class. Base classes represent the mutually exclusive and exhaustive partitioning of simple entities into cardinal groups. This is the most generic categorization of entities [9]. The definition of a base class is independent of all other classes.

Nonbase classes are formed by means of interclass connections which define the existence of classes in terms of other (base or nonbase) classes. In essence, an interclass connection comprises the membership rules of the class which it defines. Objects which satisfy the requirements specified by the membership rules are permitted entry (membership) in the class.

Interclass connections are of two major types: "subclass" and "grouping." The expression of each of these types can take on a number of different forms.

The subclass connection, as its name implies, provides for the representation of the situation where members of one class form a subset (not necessarily proper) of the members of an existing class. Membership of an entity in a subclass can depend on:

1. User specification

2. The value of an attribute of the entity

3. Membership of the object in other classes (set operator defined)

4. Service of the object as an attribute value relating to another object of a different class

The grouping connection provides a means by which a class can be partitioned to form a number of homogeneous higher (abstract) level groups. The semantics of this partitioning correspond roughly to the notion of generic decomposition, or specialization [SMIT77b] (refer back to chapter 5). The assignment of members of the underlying class into the grouping class need not be disjoint.

Each attribute used in the SDM has assigned, as part of its specification, an "applicability." The applicability is used to distinguish whether an attribute applies to the members of the class or to the class itself. In the former case, each member of the class holds a particular value(s) on the attribute. In the latter case, the entire class holds one and only one value. This single attribute can be related to the set of class member objects, however. Here, the class attribute corresponds to what is commonly referred to as an aggregate attribute [10].

N-ary associations among entities can be represented in the SDM by specifying interrelationships among the attributes of the entites. For example, a binary link between members of different classes can be established by defining that a member attribute of one class is the inverse (see [SENK75]) of a member attribute of another class. Alternatively, there can be specified a "matching" of entities from different classes according to the values of certain attributes.

The SDM provides a formalized facility to allow for the specification of derived member attributes. The facility is based on the inclusion of ten derivation primitives through which the values of member attributes

can be derived in terms of information contained elsewhere in the database. A set of attribute definition rules accompany the derivation primitive to insure that semantic inconsistency is avoided.

Eight additional derivation primitives allow for class attribute values to be derived from existing database information. Two of these primitives can be considered as aggregate functions in that their values are derived in terms of intra-class member characteristics. The remaining six primitives deal with class attributes of other classes.

Because a token entity can be a member of many classes in the SDM, a formal set of attribute inheritance rules are required. These rules specify how an entity, viewed in terms of its membership in one class, is allowed to inherit attributes due to its membership in another class. Note that only member attributes are inherited since, by definition, class attributes correspond to one object only: the class which they describe.

When a subclass is defined by means of a subclass connection such that the membership rules have to do with user specification or with the value of a member attribute, the inheritance rule is simple. Here, members of the subclass inherit all member attributes of the members of the underlying (super) class.

When a subclass connection is defined in terms of set theoretic constructs, the inheritance rules which are applied are dependent on the set operator that is used to define the subclass membership rule. Specifically:

1. If the subclass is defined as the intersection of two underlying classes, the subclass members inherit all of the member attributes of each of the underlying classes

2. If the subclass is defined as the union of two underlying classes, the subclass members inherit those member attributes that are shared by both of the underlying classes

3. If the subclass is defined as the (set) difference of two underlying classes, the subclass members inherit the member attributes of the class which serve as the minuend of the difference operation

It is important to realize that inheritance of a member attribute, used in this context, refers to both meaning and value of the attribute. Thus, despite the fact that a subclass and a respective superclass may have

class attributes which hold a similar semantic, it makes little sense to consider a subclass as inheriting (in this sense of the word) a class attribute from its underlying parent class.

The above inheritance rules are an integral part of the SDM and, as such, are applied automatically upon the specification of the corresponding interclass connection. It is hoped that the strict definition and automatic application of attribute inheritance will help to aleviate some of the semantic ambiquity which can arise as a result of user defined characteristic inheritance. A consequential disadvantage of such rules is the resulting inflexibility of the definition of subgroup member properties.

A precise specification of the SDM data definition language syntax is found in [HAMM81]. SDM database operations (data manipulation facility) have not yet been defined, however. It is expected that such a database manipulation facility, when developed, will closely parallel the existing schema definition conventions. The principle of duality of procedure and schema shows a strong influence in the design of SDM.

The Binary Approach

A binary data model is one in which each node represents a single simple attribute. Arcs which span pairs of nodes represent binary associations between these objects. The semantic of an arc is context dependent. It can depict an aggregation of attributes in the formation of an entity, or it can represent a relationship between entities.

The term "binary model" does not identify any single data modeling formalism (DMF); a number of binary formalisms have been proposed. Among the best known of these are the Semantic Binary Data Model [ABRI74], and DIAMII [SENK75]. Because the Semantic Binary Data Model is quite similar to the AI semantic net based models discussed later, we will concentrate on describing DIAMII. Our description of the modeling formalism will be brief because the binary approach has limited applicability to the domain of our concerns.

The DIAMII model is fairly representative of the generic class of binary models. Like all binary data models it is based on a few simple structures which allow for concise representation of extremely complex relationships [TSIC82]. As is the case with the other

members of the binary cohort, the model is far from being user oriented.

The DIAMII methodology. DIAMII is a multilevel model -- it models the entire system from user view to data structure level. Data semantics, the construct in which we are interested, are modeled in the "information level" of the structure. This level is second only to the "end user level." The information level deals with three major system characteristics:

1. The naming of "things" (entities) [sic]

2. The description of "things"

3. The specification of user transactions

A DIAMII paradigm is built completely from elemental "facts" which describe real world things. Elemental "things" are represented by names which are categorized into groups of names. Each group has a unique name within the paradigm. Member names are unique within a group but they may be duplicated within the overall structure. Since group member sets need not be disjoint, the

identification of a specific member requires the concatination of group name with member name (i.e. GROUP-NAME/MEMBER-NAME).

A member name defines a value that is held by the object represented. The group name to which the member name belongs provides a meaning to the value. Thus we can think of member names and group names as being values and types of values, respectively. For instance, the group/member specification:

EMPLOYEE-NUMBER/12345

refers to the "thing" employee 12345.

"Things" are described through the explication of facts. A fact is represented by the existence of an association pair (pair of named binary links) between two member names. Each participant in an association plays the role of member describing place or member described place. The association is depicted by a directed link traveling from the former to the latter role.

Within an association pair, each association is the inverse of its partner. Thus, each connected member name plays the role of member describing place for one

association and member described place for the other association.

Every association is provided with a name which must be unique with respect to its group. In other words, no two association names which eminate from a given group can hold the same name.

Figure A.16 provides an example of a DIAMII fact. Like all DIAMII facts, it has two possible interpretations:

1. Employee 12345 works in the data processing department

2. The data processing department employs employee 12345

The association names of the association pair are DEPARTMENT-OF-EMPLOYEE and EMPLOYEE-OF-DEPARTMENT. Each association of the pair is the inverse of its partner.

In the DEPARTMENT-OF-EMPLOYEE association, member name EMPLOYEE-NUMBER/12345 plays the role of member described place while member name DEPARTMENT/'DATA-PROC' is the member describing. In the EMPLOYEE-OF-DEPARTMENT association the member roles are reversed. Thus EMPLOYEE-NUMBER/12345 is describing while DEPARTMENT/

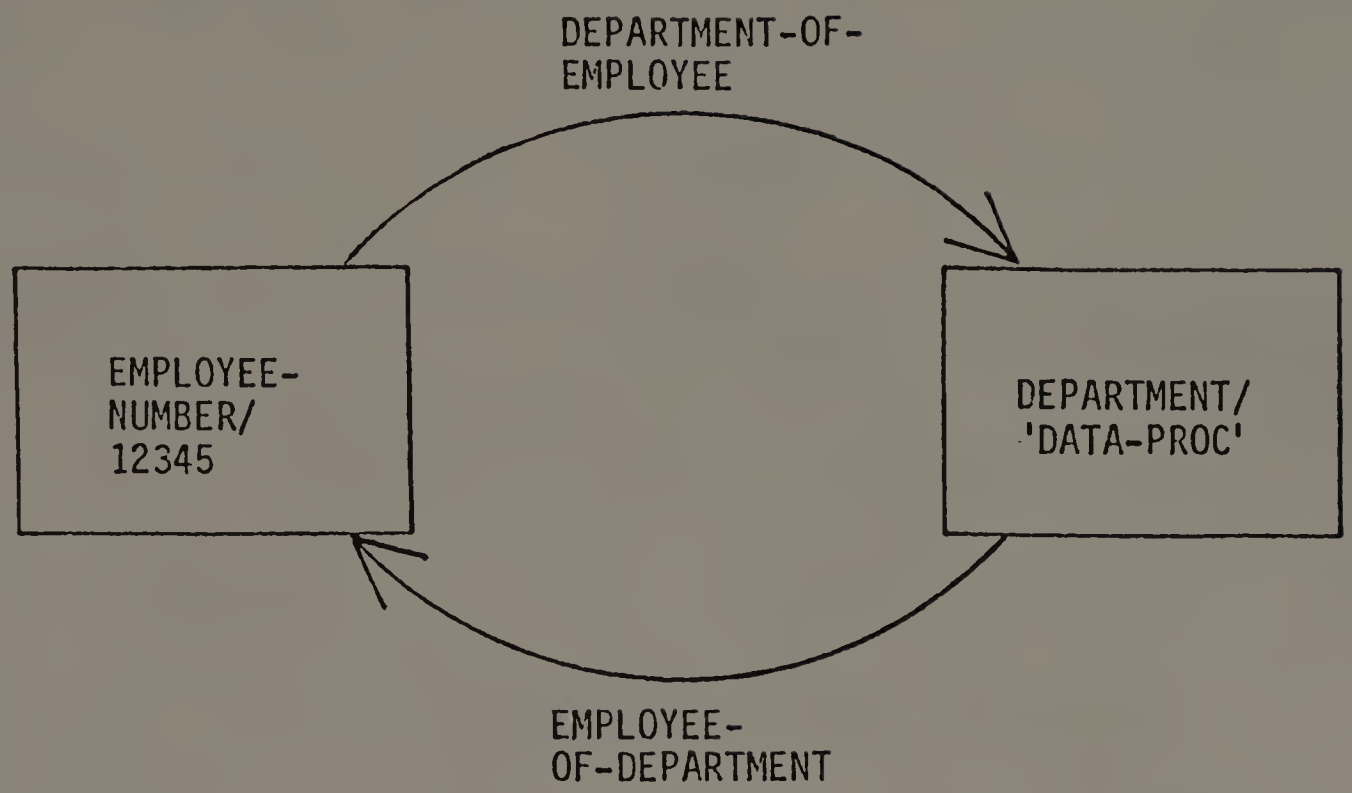


FIGURE A.16

THE DIAM II REPRESENTATION OF A FACT

'DATA-PROC' is described.

A DIAMII information level schema consists of a graph in which named groups are represented by nodes and inter group associations are represented by directed arcs. Figure A.17 provides a simple example of such a schema.

A DIAMII schema of this type represents the intension of the database in the usual way. The database extension can be conceptualized as forming a third dimension of the intensional schema. In this added dimension individual member names (values) are structured (see figure A.18).

Except for virtual information which is derivable from inference algorithms, all of the information that is contained in the system can be extracted by moving from node to node. This movement can be from member name to member name between groups, or from member name to member name within a group.

DIAMII, and binary models in general, represent an atypical approach to data modeling. The design of other DMFs shows an attempt to take information complexity from the procedures and embody it in the resulting model. This is the rationale behind the incorporation of aggregation and generalization abstraction into the constructs of a DMF.

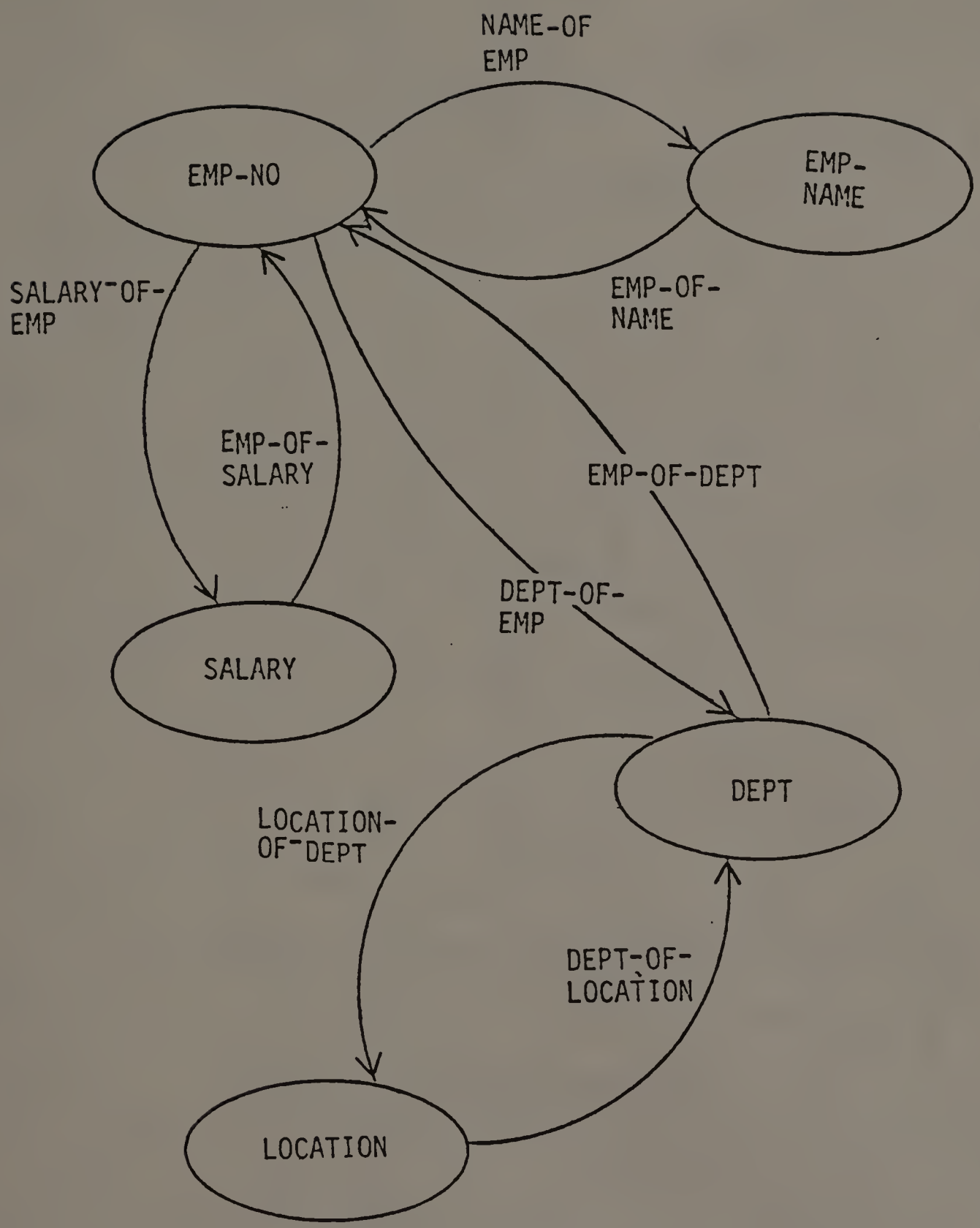

NAME-OF
EMP
EMP-NO
EMP-
NAME
SALARY-OF-
EMP
EMP-OF-
NAME
EMP-OF-
SALARY
EMP-OF-DEPT
DEPT-OF-
EMP
SALARY
DEPT
LOCATION-
OF-DEPT
DEPT-OF-
LOCATION
LOCATION

FIGURE A.17

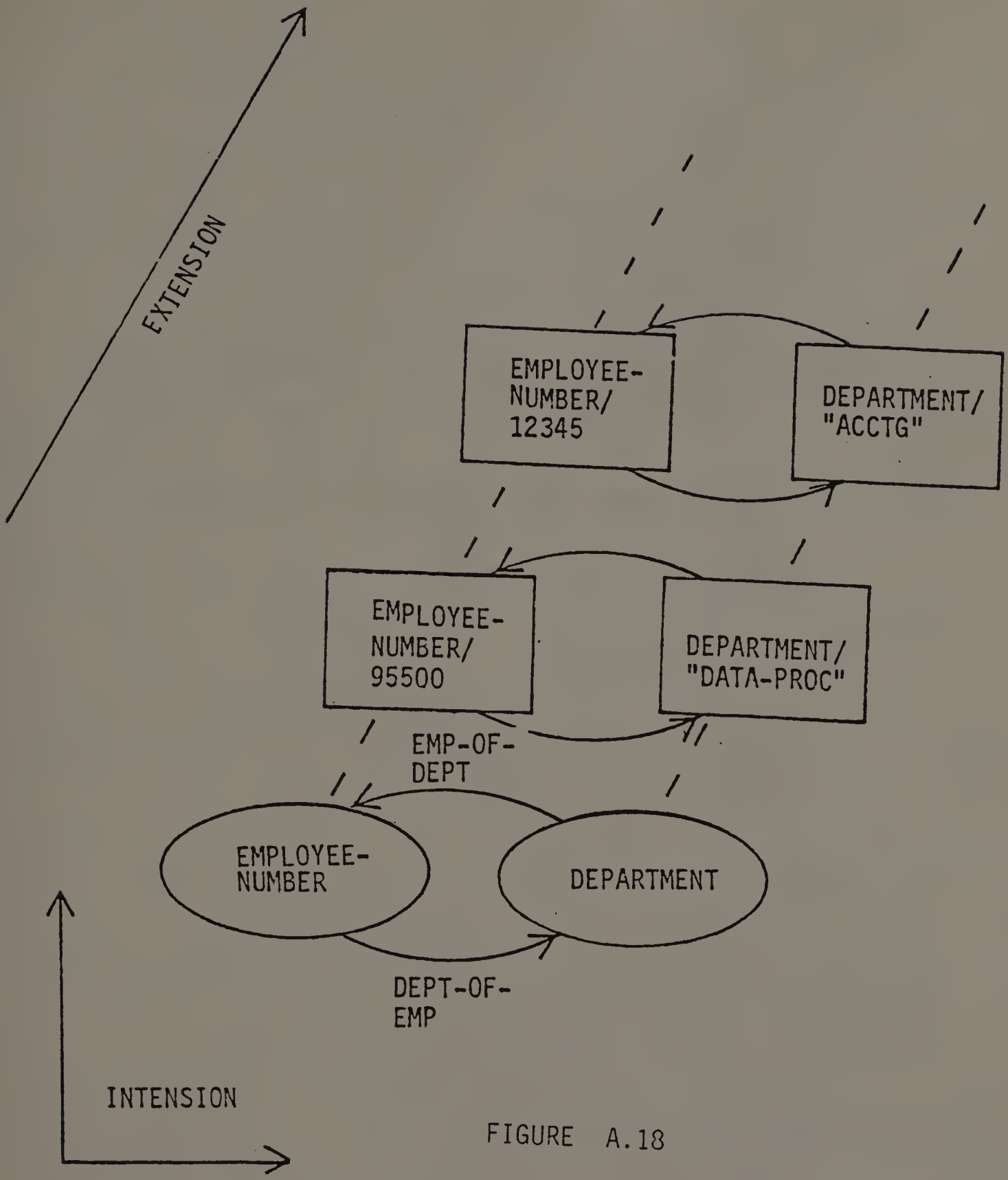

EXTENSION
EMPLOYEE-
NUMBER/
12345
DEPARTMENT/
"ACCTG"
EMPLOYEE-
NUMBER/
95500
DEPARTMENT/
"DATA-PROC"
EMP-OF-
DEPT
EMPLOYEE-
NUMBER
DEPARTMENT
DEPT-OF-
EMP
INTENSION

FIGURE A.18

In DIAMII, on the other hand, we remove all abstraction from the model structure and represent information in an elemental form. In essence, a DIAMII data model depicts the facts that can be asserted about a pool of data items. There is no single representation of an object or entity. The user must construct complex facts from the elemental binary facts. In terms of user query formulation, the user is required to "work his way through" the mass of binary assertions.

Because it provides modeling power at the cost of complexity, the value of DIAMII has been found to lie at the lower (implementation) levels of the database management system. In other words, it is useful in implementing the abstractions that are present in other more abstract models. An example of such is found in [SCHN76]. Here, the modeling constructs of the binary DIAM approach are redefined in terms of n-ary relations and used to implement a more user oriented relational data model.

AI Approaches

In recent years the emphasis placed on database applications by the artificial intelligence (AI) research community has been increasing. Furthermore, the differences in goals and methodologies between AI and data management (non-AI) efforts are decreasing [WONG77].

There can be identified five basic realms of AI work which deal with data modeling issues (according to [TSIC82]). Each of these domains revolves around the use of some form of semantic net(work). Of the five basic realms, only two are of direct concern to our work. One of the remaining three areas deals with knowledge based systems; the other two domains have to do with natural language database interfaces (see [CODD74], [HEND78], [MINK77], [MYLO75], [WALT78]).

Of those AI-based modeling realms in which we are interested, one is called the "epistemological realm" [BRAC79] while the other will be referred to as the "mathematical logical realm" (see [SCHU76]). [11].

Epistemological data modeling formalisms (DMFs) emphasize the structure of the semantic net and the inheritance rules that are implied by this structure. The models of this type most closely resemble non-AI data management products.

Semantic net nodes and arcs of the mathematical logical realm correspond to the constructs of predicate calculus. Models of this type are often called "deductive AI systems" [WONG77].

Since each of these two realms is related to our pursuit we will describe an example system from each. First, however, we feel that it is necessary to provide a brief description of semantic nets as they apply to data modeling.

A semantic net is a directed network graph. Although both nodes and edges of the graph may be labelled, only the edge labels have a semantic consequence. Node labels are for reference purposes only.

When a semantic net serves as a database schema it denotes both the intensional database (called "upstairs") and the extensional database ("downstairs") structures. Similarly, the semantic net database operation primitives mix intensional with extensional functions (recall that a non-AI schema is exclusively intensional and non-AI operations are exclusively extensional in nature). At the extensional level of the net nodes represent specific (token) things and arcs represent assertions about the things.

Unlike other database structural representations, semantic nets make use of the notion of "semantic distance" [TSIC82]. In a semantic net representation the physical proximity of two objects (nodes), measured by the number of arcs that must be spanned to form a path between them, has a definite semantic connotation. "Close" objects are more similar than are "distant" ones. In the event that the similarity of two objects would be overrated by their proximity within the net, their semantic distance can be increased by connecting them with an "irrelevancy arc."

The extensional-intensional nature of a semantic net, coupled with the strong semantic meaning embedded in its structure often results in the following situation: operations on a semantic net data model trigger the propagation of side effects throughout the net. In other words, a single simple database change is causation of a number of automatic database changes.

In non-AI database situations such multiplicity of side effects would be considered as anomalous and aberrant. Indeed, much research effort is directed towards the avoidance of such (see [HAMM75]). In the AI view effect propagation is considered as being normal, however. The data model is seen as a semantically

self-adjusting system. We feel that the intuitive pleasantness of this viewpoint is difficult to deny.

Despite the elegance and power of semantic net based AI data models, they are not without their problems. First of all, a semantic net is, in all case, expensive to design, store, maintain, and process. Furthermore, it is often noted that semantic net implementations are an example of "overkill" for any particular database application. Semantic nets may be overly complex because they include so many concepts -- they are aimed at modeling too many real world provinces. This opinion is put forth in [SU79], [HAMM81], and [TSIC82].

An epistemological based system. In this section we describe a semantic net database management system (DBMS) as presented in [ROUS75]. Many of the concepts discussed hereunder have evolved from the TORUS natural language understanding DBMS project (see [MYLO75]). We believe that these concepts provide a thorough overview of the epistemological realm.

The nodes of the semantic net are of four basic types:

1. Concepts -- depict the physical and abstract objects of the real world

2. Characteristics -- serve as modifiers of concepts, events, or other characteristics

3. Values -- are the values taken on by the above characteristics

4. Events -- emulate the actions of the real world

The graphical representation of a characteristic node includes two edges. One edge, labelled "ch" (characterize), is directed toward the object which is modified by the characteristic. The second "v" (value) edge points to the associated value of the characteristic. The characteristic node is labelled with the name of the characteristic.

Figure A.19 provides an example of a characteristic. This semantic net representation asserts "Barbara has blue eyes." Note that an alternative conceptualization of a characteristic is that of a binary relation which maps an element from its domain to its range. In the above example, viewed in this light, "barbara" is an element of

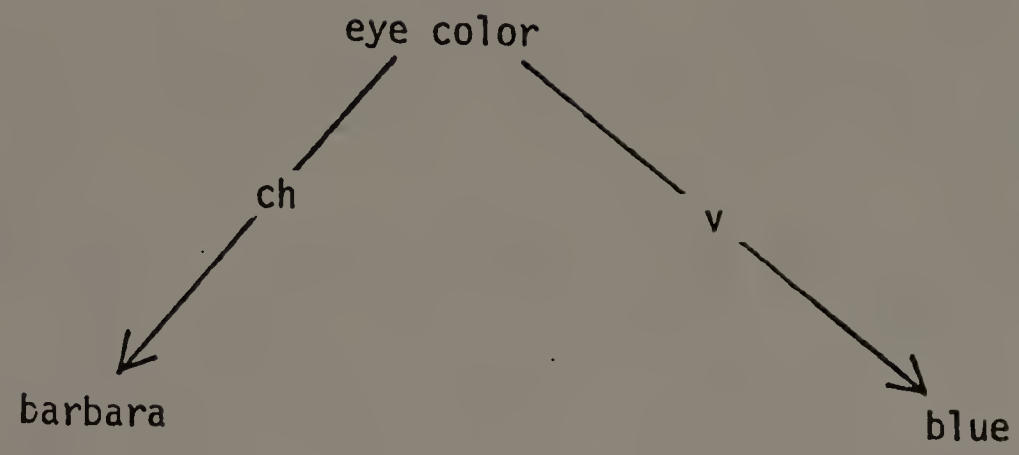


FIGURE A.19

A CHARACTERISTIC

the domain (set of objects which may be characterized) of the relation. "Blue" is an element of the range (set of possible values which may be taken on).

Events are complex objects in that the representation of a single event consists of a number of nodes and edges. One node, the event node, depicts the event itself. The remaining nodes of the event represent objects which play specific roles in the event [12]. These roles are specified by labelled edges which are directed from the event node to the role playing object. Common roles are affected (aff), agent (a), destination (d), instrument (i), object (o), result (r), source (s), and topic (t).

An example of a semantic net representation of an event is given by figure A.20. The "fact" that is asserted by this representation is "CDC supplies executive software to UMass." This figure illustrates that objects are allowed to play more than one role in an event. The object "cdc," for instance, is both the source of the object, and the agent in the example event.

When concepts are engaged in complex situations such as events and the like, their characterization often requires qualification. In the above network, for example, the concept "executive software" can be

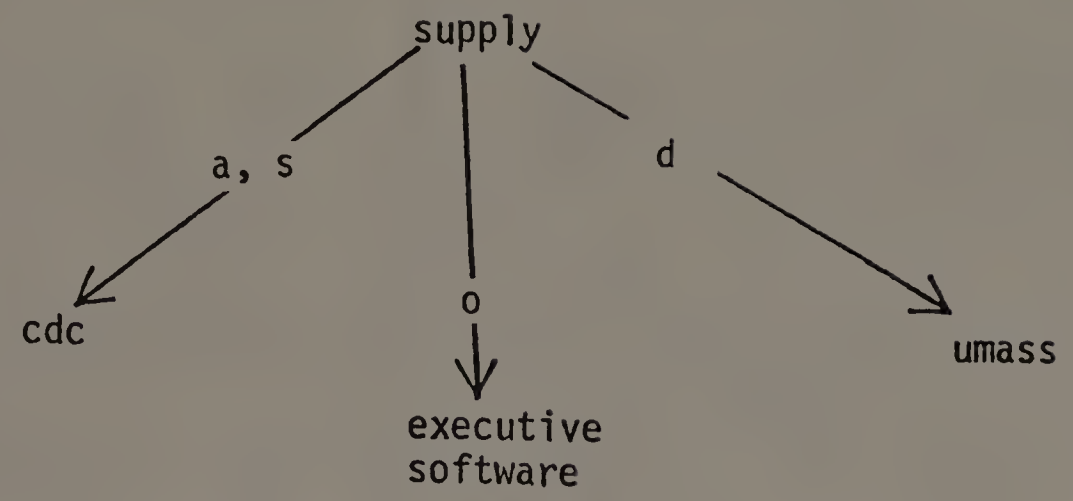


FIGURE A.20

AN EVENT

characterized by a particular price. The price is not a function of the concept alone, however. It is determined by the event (supply) in which the concept plays the object role. Thus, the characterization of the concept must be specified "in terms of," the situation. Mathematically, we must map from a two-dimensional cross product domain to a range of values.

In this instance, the "wrt" (with respect to) edge provides the appropriate qualification. Figure A.21 shows how the executive software concept is characterized with respect to the specific supply event in which it plays a role.

In an AI model of this type the single semantic net describes both the intension of the database and the extension of the database. Intensional (upstairs) nodes are labelled in capital letters. Extensional (downstairs) nodes have lower case labels. Note that the prior examples have all been extensional database representations.

The upstairs of the net serves as a generic template through which the downstairs objects are structured. The two representations are connected by means of the "E" (example of) labelled edge which is directed towards the extensional token. Figure A.22 provides an illustration

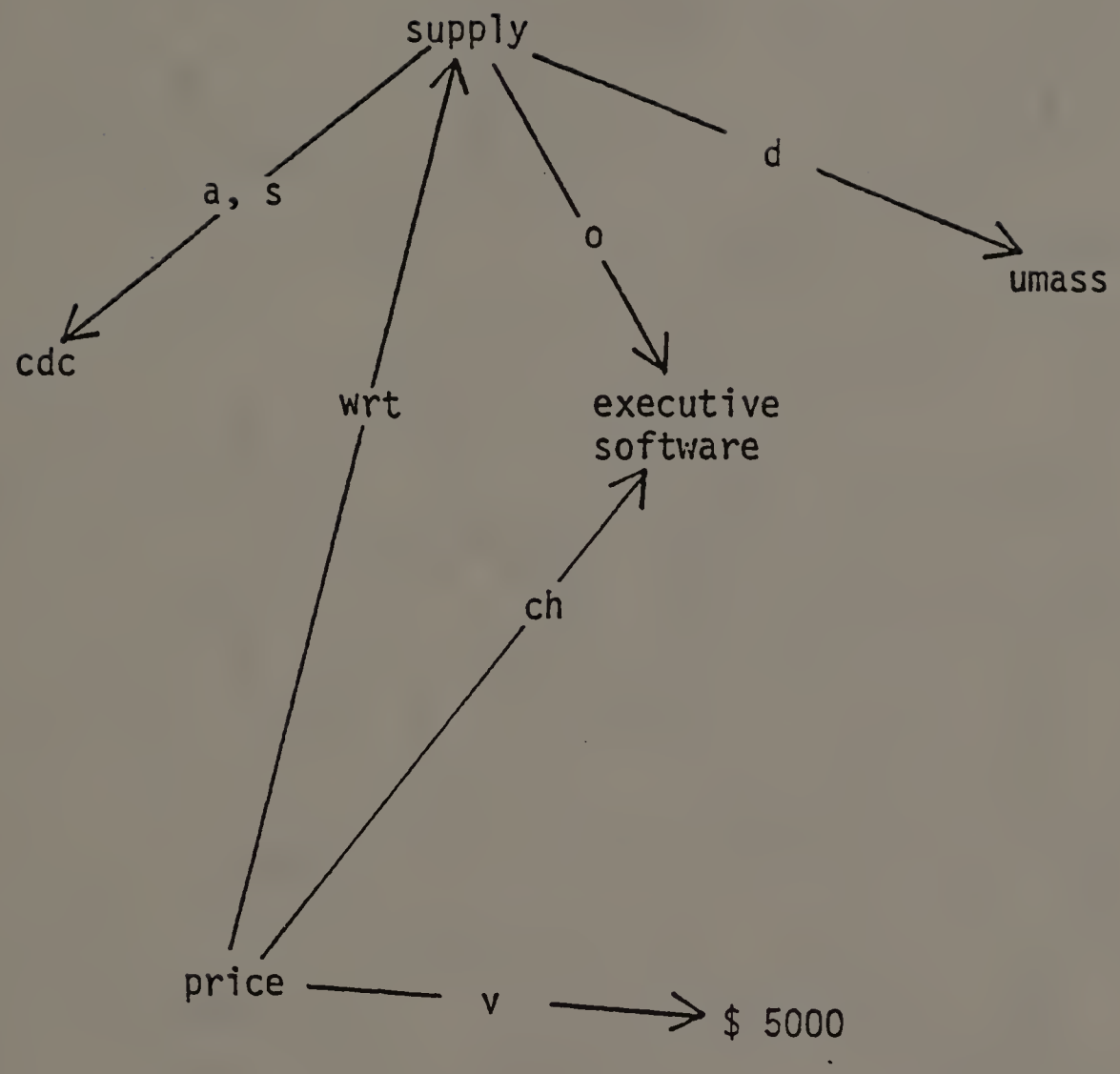


FIGURE A.21

QUALIFIED MODIFICATION OF A CONCEPT

of how the upstairs and downstairs structures are connected.

The semantics of causation are captured through utilization of the "effect" and the "prereq" (prerequisite) edges. As an example, consider the following assertion: "A part must be shipped to a customer before it is received by the customer." In essence, this states that shipment is a prerequistie of receipt. Figure A.23a illustrates the semantic net (upstairs) representation of this assertion.

Now consider the assertion "The occurrence of an order event will bring about the occurrence of a supply event." In other words, order effects supply. Figure A.23b illustrates this.

"Knowledge chunks," called scenarios, constitute the semantic net structure. Each scenario consists of a set of interrelated concepts, events and values that have a distinct semantic foundation. The structure of a scenario forms a "pattern template" which, when matched with another structure, provides meaning and understanding of the latter to the system.

The individual pieces of knowledge, represented by scenarios, are organized according to three basic axes, or dimensions, of the overall semantic net. Movement along

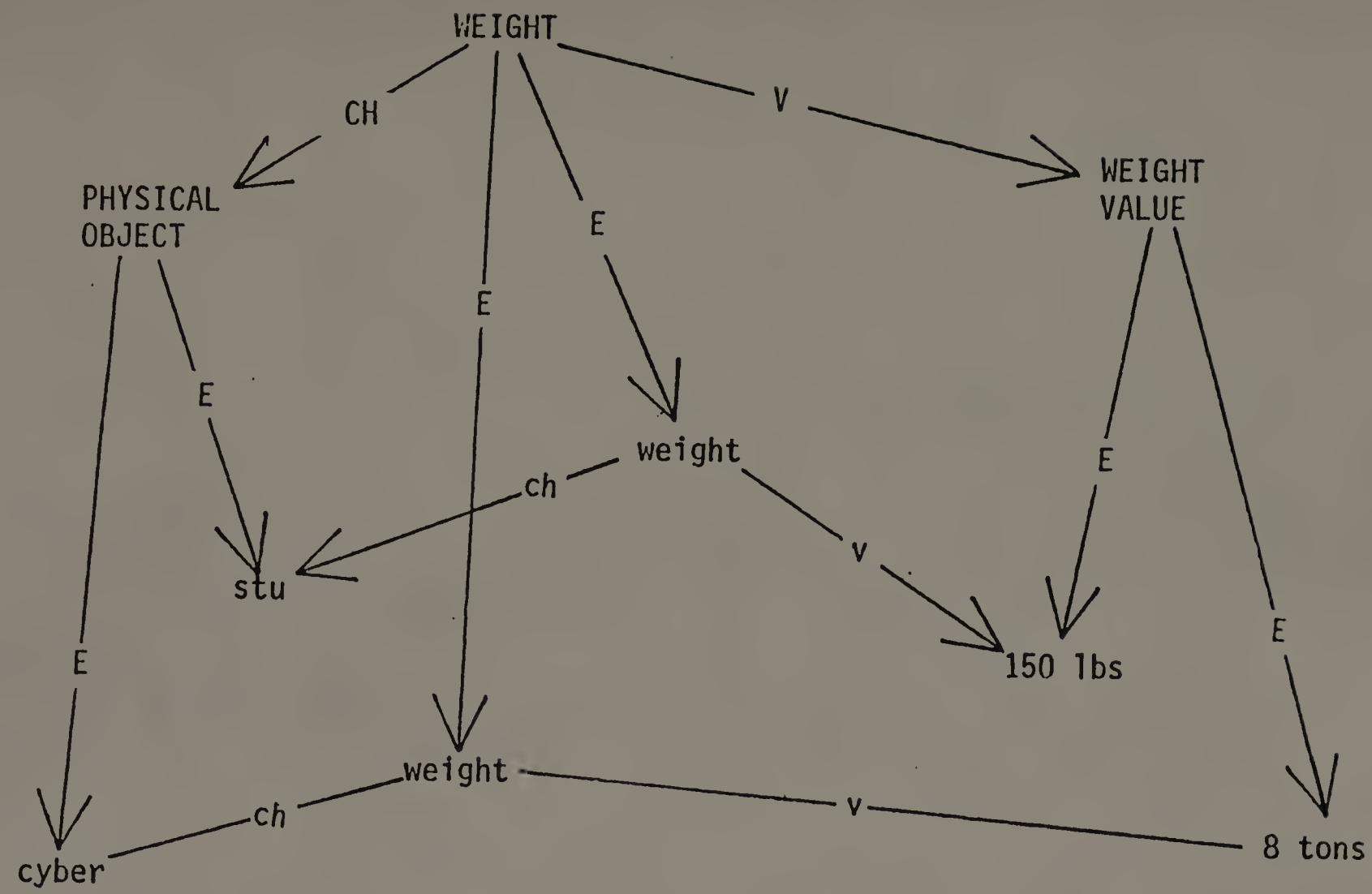


FIGURE A.22

UPSTAIRS AND DOWNSTAIRS

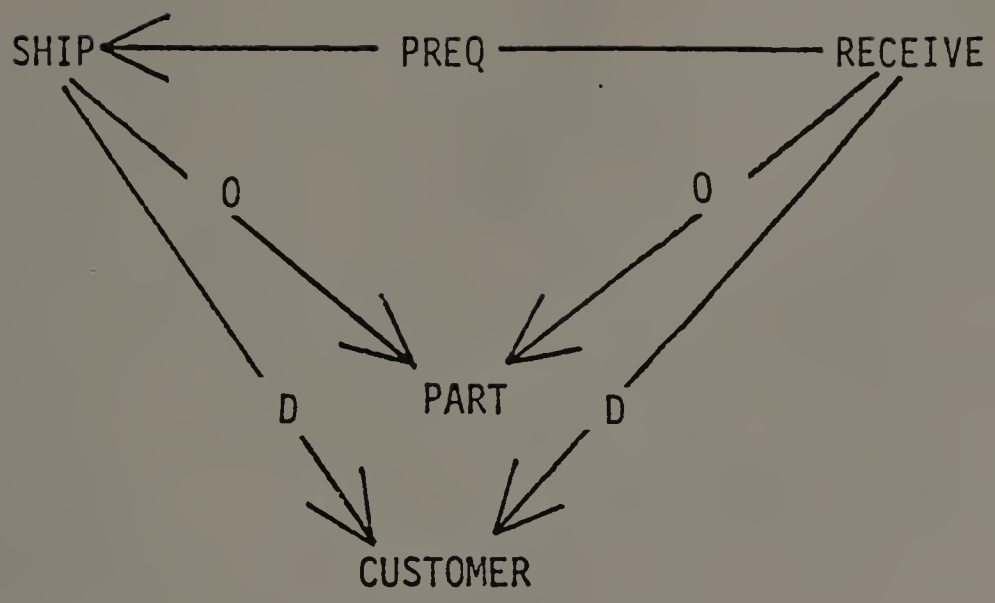


FIGURE A.23a

FIGURE A.23b

each dimension has definite semantic implications. The first axis, which is based on the notion of the subset relation, is called the "SUB" axis. Scenarios which appear lower on the SUB axis are subsets of those that appear higher on the axis.

If an event E is a subset of another event E', then all concepts which play roles in E are subsets of the respective concepts that play roles in E'. The semantic properties of the super concepts (roles of E') are inherited by the subconcepts accordingly.

The second axis, called "DEF," is definitional in that movement along the axis provides varying degrees of detail about characteristics, concepts, and events [13]. For example, the concept "CDC COMPUTERS" is located below the concept "COMPUTERS" with respect to the DEF dimension.

The final dimension through which the scenarios are organized is the "PART" dimension. Concepts which appear lower on this axis are components of the higher concepts [14]. Thus "DEPARTMENT" is located below "COMPANY."

The inclusion of a fourth axis to represent a "TIME" dimension has been considered. Through the use of such a dimension each scenario would be assigned a "period of applicability." As a result, the net would obtain a

knowledge of its own developmental history and, accordingly, of the history of the universe of discourse which it represents.

The intelligent database schema "understands" user-system dialogue by matching the (semantic) graphic structure of the dialogue to the structure of its internal scenario templates. For each statement and query that enters the system, a graph is constructed. This graph is then processed through a graph-fitting algorithm which moves the input graph as far down along the net axes as is possible. Additional inferences are made through the use of context -- the existing understanding gained from the previous dialogue.

At a lower structural level, the system is implemented as a relational database. There are four basic types of relations which have a natural correspondence to three of the basic node types of the semantic net. These node types are concept, event, and characteristic. Value nodes of the net are not represented as relations for obvious reasons.

A unique feature of the DBMS is that, despite its relational foundation, it does not depend on the use of keys. As a consequence, the user of the system is allowed to hold a more "natural" conceptualization of the concepts

represented. For example, employee tokens can be thought of as persons rather than as unique employee numbers.

Because the relational database is structured within a semantic framework, the relational database operation primitives (selection, union, intersection, difference, division [CODD70]) require modification. Five corresponding semantic operators, all of which are set theoretic in nature, have therefore been defined (see [ROUS75] for details). Each of these semantic operators is directly applicable to the SUB axis of the semantic net because said dimension is based on set theoretic containment.

A consequence of using semantic database operators vis. the traditional strictly mathematical operators is that the system obtains the capability to assess the "legality" of database transactions. There can be defined a "measure of strangeness" of an operation invocation. The value of said measure is a function of the relative (according to existing context) height which must be achieved on the SUB axis in order to fit a dialogue scenario pattern.

As the height of the pattern match increases, the less related are the entities involved in the operation to the present contextual state. "Strange" operations can

thusly be identified by the system. Identification of such anomalies can raise questions (on the part of the system) as to the credibility of the user and/or the transaction.

A mathematical logic based system. Deductive AI systems are based largely on deductive rules which take the form of predicate calculus quantified statements (extensive discussions of predicate calculus are found in [WONG77] and [BONC81]). The major objective of research on such systems is to "develop a system from which one can deduce facts which are implicit within the database" [MINK77, p. 108].

Deductive AI systems are rooted in a many sorted logic vis-a-vis first order logic. In first order logic all predicates and arguments are elements of a single universal domain. Consequently, any combination of predicates and terms that is a well formed formula (wff [BONC81]) is valid (well formedness is essentially syntactic correctness). For example, if the terms "blue" and "banana," and the predicate "FATHER(x,y)," are elements of the universal domain; then the following wff is valid:

FATHER(BLUE,BANANA)

The "fact" represented by this wff is "blue is the father of banana" which is an obviously nonsensical assertion (from [MINK77]).

Such anomalous assertions can be avoided by using a many sorted logic. Here, terms and quantifiers are restricted to range over different domains. Hence, for any given predicate, its arguments can be restricted to come from domains which constrain the resulting assertion to be semantically meaningful. The arguments of the predicate FATHER(x,y), for example, would be restricted to be elements of the domain "person" vis. the domain "color" or "fruit."

A wff that makes the same assertion about all members of a domain (conjunctive) can be expressed in terms of the universal quantifier:

∀ (for all)

Disjunctive assertions which refer to all members of a

domain can be expressed in terms of the existential quantifier:

∃ (there exists)

The theorem proving methodology of deductive AI systems requires that wffs be converted to clause form: a conjunction of clauses which is quantifier free [15]. The procedure through which the existential and universal quantifiers are removed, which is based on the use of Skolem functions and the resolution principle, is described in [BONC81], [MINK77], and [WONG77].

In the remainder of this section we will describe briefly the mechanics of the MRPPS 3.0 [MINK77] -- a (experimental) deductive relational database system. We feel that this system, which provides one of the few examples of a general purpose deductive database system, is representative of the mathematical logical realm of AI data modeling.

The MRPPS model of knowledge is stored in the semantic network [16]. The overall network comprises four components:

1. The semantic graph

2. The two databases (see below)

3. The dictionary

4. The semantic form space

The semantic graph denotes relationships (disjointedness, inclusion, overlap) among semantic categories in a network graph structure. Figure A.24 provides a simple example of such a graph. From this particular structure we can draw a number of implications:

1. "A person who is a mother is a female person"

$$MOTHER(x) \rightarrow FEMALE(x)$$

2. "A person who is female is not a male person"

$$FEMALE(x) \rightarrow \sim MALE(x)$$

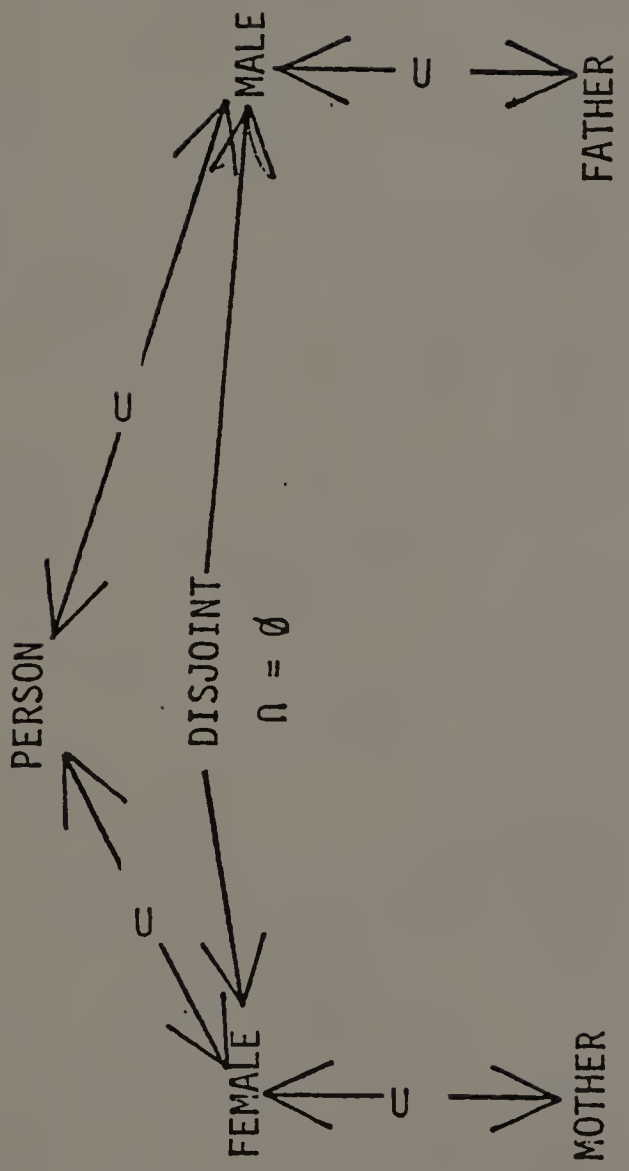


FIGURE A.24

AN EXAMPLE SEMANTIC GRAPH

3. Accordingly, then, "A mother is not a male person"

$$MOTHER(x) \rightarrow \sim MALE(x)$$

As in all deductive AI systems the MRPPS database is a twofold entity. It comprises the intensional database (IDB) and the extensional database (EDB). The former stores clauses only; the latter stores assertions (note that assertions are really instantiated clauses).

Consider the following clause that could be stored in the IDB:

$$FATHER(x,y) \wedge HUSBAND(x,z) \rightarrow FATHER\text{-}IN\text{-}LAW(x,z)\}$$

This representation translates into:

"x is the father of y and ($\wedge$)
y is the husband of z implies ($\rightarrow$)
x is the father-in-law of z"

The clause is intensional because the terms x, y, and z are variables.

Now, if the EDB stores actual data (assertions) about fathers and husbands, then the IDB "stores" virtual data about fathers-in-law. The distinction between the IDB and the EDB is therefore related to that between direct and derived data. The contents of both databases are stored as relations.

The dictionary lists the semantic category of each relation in the database. This provides the bridge between an object and its "meaning." Recall that the semantic categories to which objects are linked exist in the highly structured semantic graph.

The final network component, the semantic form space, defines the semantic constraints which are imposed on predicates and arguments. Obviously, this invokes the many sorted logical structure upon which such systems are based.

The database user queries the system by specifying a conjunction of clauses [17]. Clauses are defined in terms of a literal template and a substitution set. The literal template demarcates the structure that the terms of the clause will take on. The substitution set specifies the values of the predicate and its arguments.

The following literal template defines a predicate (alpha) with two arguments (x1, x2):

$$(\alpha,x1,x2)$$

The query specification

$$\{(\alpha,x1,x2)\ [FATHER/\alpha,person/x1,JOE/x2]\}$$

asks "Who is Joe's father?" Similarly,

$$\{(\alpha,x1,x2)\ [MOTHER/\alpha,person/x1,JOE/x2\}$$

queries "Who is Joe's mother?" Note that many queries can have the same literal template structure.

Queries are answered by the deductive search mechanism which has the ability to deductively form new clauses from sets of input clauses. The semantic form space and the semantic graph provide the deductive search mechanism with knowledge as to which queries are

unanswerable.

Consider, for example, the following query:

$$(\alpha,x1,x2) \wedge (\beta,x1,x2)\ [FATHER/\alpha,MOTHER/\beta,person/x1,JOE/x2]\}$$

This asks "Who is both the mother and the father of Joe?" The deductive search mechanism would reject this query since:

$$FATHER(person) \rightarrow MALE(person)$$

$$MOTHER(person) \rightarrow FEMALE(person)$$

$$MALE(person) \rightarrow \sim FEMALE(person)$$

The database (IDB and EDB) is accessed through the knowledge base index structure. This consists of three hierarchically organized parts:

1. The literal size and sign list

2. The literal template tree structure

3. The inverted index file

The literal size and sign list points to that template tree structure which relates to the templates of a given size (number of terms) and sign (negated or not). Each template tree structure is a binary tree which is the basis of a pattern matching process.

Proceeding down a branch of the template tree relates to the matching of the structure (argument pattern) of a clause. Each terminal node of the tree points to the constraints (form space) that refer to the clause structure, and to the appropriate section of the database. Inverted indexes drive the search at the database level.

The MRPPS system output is available in three different forms:

1. Symbolic (predicate calculus based) output

2. Written natural language output

3. Natural language voice output

It is important to keep in mind that the MRPPS is an experimental prototype DBMS.

FOOTNOTES

[1] The ultimate purpose of normalization is the same as that of E-R modeling: avoidance of semantic ambiguity. The resulting representations may be quite different, however.

[2] It should be noted that the semantic complexity of ternary and higher order relationship sets often results in a representation that is difficult to comprehend [TSIC82].

[3] Note that the meaning of an E-R attribute is quite different from that of its relational homonym. The E-R term refers explicitly to the mapping from object to domain.

[4] In general, an entity relation and a database relation are not synonymous. Entity relations are not, in all cases, required to have unique keys as are database relations.

[5] This corresponds to the "dependency constraints" as described in [WONG77].

[6] The RM/T model [CODD79] includes "nested" characteristics that are similar to the complex characteristics of the BSDM. The RM/T constructs are called "characteristic molecule types."

[7] Most network representations guarantee that the deletion of an object causes the deletion of all connective links to that object. Such representations therefore have a consistency rule implied by their structure. This rule is called the "existency constraint" [WONG77].

[8] RM/T allows for the definition of "unconditional successors" and "alternative successors" to capture the meaning of the time ordering of events. The former notion most closely resembles the SAM "CF" concept. The latter notion is slightly more powerful in that it specifies the occurrence of one of a number of possible events.

[9] An interesting feature of this model is that the creation of the most generic abstract object is considered as the lowest level of abstraction. Under the usual Smith and Smith configuration, such an object is placed at the top of the (generalization) abstraction hierarchy and therefore represents the highest level of abstraction.

[10] A class attribute is quite similar to an attribute of a "metaclass" as used in an AI based model.

[11] Epistemology is the study of knowledge.

[12] Use of the term "role" by the artificial intelligence community is usually restricted to data management applications. The more general term is "class." Besides providing a uniform method for representing the intension of an event, case representation provides a direct bridge between the syntactic structure of the natural language representation of a fact and the semantic meaning of a fact.

[13] Both the SUB construct and the DEF construct relate to the notion of generalization abstraction. In most AI applications these two dimensions are collapsed to form a single IS-A dimension (see [WONG77]).

[14] The PART construct captures the concept of aggregation abstraction. A more commonly used AI label for this dimension is PART-OF (see [WONG77]).

[15] Clauses are negations and/or conjunctions of atomic wffs.

[16] The semantic network, as described here, represents a more comprehensive object than is commonly used. The first component of this network, the semantic graph, most closely resembles a traditional AI semantic net.

[17] Disjunctive sentences can be converted to fully conjunctive clauses through the proper use of negation (~).

A P P E N D I X B

GEOGRAPHIC CONCEPT DEFINITIONS

The following geographic concept definitions and descriptions are taken directly from "Geographic Concept Definitions" (pp. 53-70), Census of Population and Housing, 1980 -- Summary Tape File 1: 1978 Richmond Dress Rehearsal Technical Documentation / prepared by the Data User Services Division, Bureau of the Census -- Washington: The Bureau, 1980 ([CENS80]). Some format and punctuation conventions used below are therefore slightly different from those employed elsewhere in our work; for this minor inconvenience, we apologize.

Block. Normally a well-defined rectangular piece of land, bounded by four streets. However, a block may also be irregular in shape or bounded by railroad tracks, streams or other features. Blocks, by definition, do not cross the boundaries of counties or census tracts. They may cross place boundaries and the boundaries of minor civil divisions. When blocks cross place boundaries and, in 20 States (. . .), when they cross MCD boundaries, the blocks are split and statistical summaries are presented for the parts.

Census data will be tabulated by block in all urbanized areas (UA's) and, in many cases, somewhat beyond the final UA boundaries. The data will also be tabulated by block in incorporated places with 10,000 or more inhabitants outside UA's and in additional areas which contracted with the Census Bureau for the collection of block statistics. Places outside of UA's are included in the block statistics program if they met the 10,000 population criterion in the 1970 census, official Bureau estimates through 1976, or a special census on or before December 31, 1977. Block coverage for qualifying places is within boundaries as of January 1, 1980. Five States contracted for the preparation of block statistics covering their entire territory, both urban and rural: Rhode Island, New York, Virginia, Georgia, and Mississippi. (. . .)

A block is identified by a 3-digit code which is unique within census tract or, where tracts do not exist, block numbering area. Since separate summaries are provided for the parts of a block split by a place or, in 20 States, an MCD boundary, tape users may need to specify the place or MCD code to retrieve data for a block. Blocks are defined on detailed census maps: Metropolitan Map Series, Vicinity Map Series, place maps, and county

maps. The extent of block statistics coverage is reflected on maps by the presence or absence of the 3-digit block number.

Census blocks are normally compact cohesive units, but there are important exceptions. For example, in some suburban areas, development is clustered around cul-de-sacs. In these areas a census block may be fairly large since only those streets that serve as the perimeter of an enclosed area are treated as block boundaries. Also, in rural areas blocks may include many square miles, depending on the frequency of roads and their intersection with rivers, mountain ridges or other physical features.

On census maps, when a block boundary ignores a minor physical feature, such as railroad tracks, a "fishhook" (. . .) across the feature indicates that the block includes area on both sides of the feature. Alternatively, the separate parts of such a block will have the same block number followed by an asterisk.

The maps used for enumeration activities were, of necessity, obtained one or more years prior to the census and therefore do not reflect recently constructed streets, if any. Block statistics observe only those block boundaries shown on the maps.

It is estimated that statistics will be collected for over 2.5 million blocks in the 1980 census. Block statistics are included in PHC(1) Block Statistics reports and in file B of Summary Tape File 1 (STF 1B).

Historical comparability: In 1970 block statistics were prepared for urbanized areas and some additional contract areas. Unlike in 1980, they were not prepared for places of 10,000 population or more outside urbanized areas, unless done under contract. More than 30 percent of the areas for which block statistics will be available for 1980 did not have block statistics in 1970.

Some blocks defined for 1970 will have new boundaries in 1980, primarily those on the edges of urbanized areas and other areas of new development where the street patterns have physically changed. To help the user notice a change wherever a block has been redefined by splitting or other adjustment, the 1970 block number will generally not be reused. In many areas, however, block boundaries and numbers will be the same in 1980 as in 1970, except for a few areas where blocks were renumbered by local GBF/DIME coordinating agencies to define more desirable block groups.

Block group (BG). A combination of census blocks which is a subdivision of a census tract or block numbering area (BNA) and which is defined in all areas where block statistics are collected. (In areas where blocks are not identified, enumeration districts substitute for block groups as tabulation units.)

Block groups are defined within county and tract or block numbering area. They may be split by the boundaries of other higher level geographic entities recognized in the census, including places, minor civil divisions or census county divisions, congressional districts, and Indian reservations. When this occurs, statistical summaries (data records) are provided for each component or part.

Block groups are not outlined on census maps, but are defined as that set of blocks sharing the same first digit within a census tract or BNA. For example, Block Group "3" within a particular census tract would be defined as all blocks numbered between 301 and 399. In practice, the numbering would rarely go above 350 and would involve substantially fewer than 50 blocks, since gaps are occasionally left in the numbering, e.g., one block might be 312 and the next 316.

Since block group summaries observe higher level boundaries, users should carefully study census maps to note the presence of place, MCD or CCD boundaries which may split block groups. Congressional district boundaries are not shown on census maps; as a result, a block group may be split but the boundary will be undefined.

Block group summaries observe some boundaries which are ignored in block statistics (specifically, census county divisions and, in 10 States, minor civil divisions). As a result, it may be necessary on rare occasions to add two block group components together to match the sum of blocks in the same hundreds series.

It is estimated that statistics will be prepared for about 180,000 block groups. Block group data, together with data for enumeration districts, appear on STF's 1A, and 3A, and corresponding microfiche. There are no printed data for block groups.

Historical comparability: In areas where block groups were tabulated in 1970, most 1980 block groups will be the same as their 1970 counterparts, with exceptions occurring primarily in areas where tract boundaries have changed. In addition, block group parts, created when block groups are split by the boundaries of other higher level areas, will also change if such boundaries have been changed.

Many areas with block groups in 1980 had enumeration districts in 1970, a change occasioned in part by the expansion of the block statistics program, and in part because ED's were used for tabulation purposes in 1970 instead of block groups in some blocked areas. Where block groups have replaced ED's, there will be little comparability between 1970 ED's and 1980 block groups.

Block numbering area (BNA). Areas defined for the purpose of grouping and numbering blocks in blocked areas where tracts are not defined. BNA's do not cross county boundaries. They are identified by census tract-type numbers ranging from 9901.00 to 9989.99 which are unique within a county. While BNA numbers are similar to census tract numbers, BNA's are not census tracts and are not included in STF 2 or 4.

Block numbering areas may be split for tabulation purposes by the boundaries of places, Minor Civil Divisions (MCD's), and Census County Divisions (CCD's). STF's 1A and 3A present statistical summaries for the component parts of BNA's created when BNA's are split by the boundaries of places, MCD's, and CCD's. BNA summaries in STF 1B and PHC(1) Block Statistics reports recognize the boundaries of places, as in the other files, but CCD boundaries are ignored and MCD boundaries are recognized only in 20 States (. . .).

Historical comparability: While BNA's were also used in previous censuses, any historical comparability is only coincidental.

Census County Division (CCD). Subdivisions of counties, established in 20 States (. . .) which do not have MCD's suitable for reporting census statistics (i.e., the MCD's have either lost their original significance, are very small in population, have frequent boundary changes, and/or have indefinite boundaries). CCD's are established cooperatively by the Census Bureau and both State and local government authorities. CCD's are generally defined by boundaries that seldom change and can be easily located, such as roads, rivers, powerlines, etc.

Census county division boundaries are represented on all detailed census maps. In addition, CCD outlines appear at a small scale on maps published in PC(1)-A and HC(1)-A reports. CCD's, in alphabetic sequence, are assigned unique, incremental 3-digit numeric codes within counties.

There are approximately 5,500 CCD's defined for the 1980 census. Statistics for all CCD's appear in STF's 1A, 2B, 3A, and (under tentative plans) 4B, in PC(1)-A and -B, and HC(1)-A reports.

Historical comparability: CCD's are now defined in one fewer State than in 1970 -- North Dakota returned to the use of its MCD's (townships) for 1980. In 1960 there were 18 CCD States. In the past, cities with 10,000 or more

> inhabitants generally were defined as separate CCD's. When these cities annexed territory, CCD boundaries also had to be adjusted. For 1980, many of these CCD boundaries were revised to conform with census tract boundaries where tracts exist, and permanent physical features elsewhere, in an attempt to minimize future CCD boundary adjustments.

Congressional District. These 435 areas are defined for their respective States by State legislatures for the purpose of electing persons to the U.S. House of Representatives. Congressional districts observed for the 1980 census are as designated for the 96th Congress; this designation has been in effect since the 94th Congress (1975-1976), with one boundary change in Tennessee which took effect with the 95th. Data summaries from the 1980 census appear only in STF 1A. Congressional districts for the 98th Congress (1983-1984) will be defined by State legislatures shortly after 1980 population counts become available.

Small scale maps of congressional districts appear in the Congressional District Data Book. Congressional district boundaries are not shown on detailed 1980 map series.

Historical comparability: 1970 census data are available for congressional districts as defined for the 94th - 97th congresses in the Congressional District Data Book, except for the Tennessee change noted above.

County. The primary political and administrative divisions of States. In Louisiana such divisions are called parishes, and in Alaska 23 boroughs and census areas are treated as county equivalents. Several cities (Baltimore, Maryland; St. Louis, Missouri; Carson City, Nevada; Columbus, Georgia; and a number of Virginia cities) are independent of any county organization and thereby constitute primary divisions of their States and are treated the same as counties in census tabulations.

County boundaries are shown on all census maps. A 3-digit Federal Information Processing Standards (FIPS) county code identifies each county uniquely within State. Counties are numbered in alphabetic sequence, with independent cities numbered separately at the end of the list.

There are 3,137 counties and county equivalents being tabulated for the 1980 census. County tabulations appear in STF's 1 through 4, and in PC(1)-A, -B, and -C; HC(1)-A and -B; and PHC(3) reports.

Historical comparability: A number of changes have occurred to county boundaries since 1970, mostly as a result of the creation of new independent cities or annexations by independent cities in Virginia. A new set of county equivalents has been defined for Alaska (boroughs and census areas) which in some cases differ considerably from the divisions recognized for 1970. In addition there are minor changes in the list of counties for South Dakota, Georgia and Hawaii.

Division, (Census Geographic). Census geographic divisions are nine groups of States which are subdivisions of the four census regions. (. . .) Divisions are identified by a 1-digit code which is also the first digit of the 2-digit Census geographic code for each State in the division.

Historical comparability: Census divisions have remained largely unchanged since the 1910 census.

Enumeration District (ED). An area used in the 1980 census for collection activities and as a tabulation area where block statistics are not prepared. Enumeration districts do not cross the boundaries of any other legal or statistical area, including MCD's, CCD's, places, counties or congressional districts. Because of these

constraints they vary widely in population size, although they do not generally exceed a population of 1,600 in areas where the census is taken by mail, or a population of 1,000 in areas where the census is taken by conventional enumerator canvassing. The population limits are designed so that an ED represents a reasonable workload for a single enumerator. About 1,000 areas in 47 States participated in a program for local definition of ED's. In the areas where they are reported, ED's are the smallest available unit of census geography.

Enumeration district boundaries are shown on MMS/VMS, place and county maps in areas where there are no block numbers. ED's are identified by a four-digit number which may be followed by a one-character alphabetic suffix. The suffix is used to identify subdivisions of ED's made during data collection activities where the original ED proved to be too populous for an efficient work unit, or to accommodate a revision to a place or other boundary made too late to be reflected on the maps. An ED number may also have a prefix indicating that the ED is of a special type (e.g., an Indian reservation), but the prefix is not necessary for identification of the ED. ED numbers do not repeat within a county.

It is estimated that statistics will be prepared for about 120,000 enumeration districts. ED data, together with data for block groups, appear on STF's 1A and 3A and corresponding microfiche. In addition, ED data appear on STF 1B to complement the summaries for blocks. There are no printed data for enumeration districts.

> Historical comparability: Many areas which were covered by enumeration districts in 1970 are summarized in terms of blocks and block groups for 1980. In some cases it may be possible to add up blocks to approximate the 1970 ED, based on detailed comparison of 1980 and 1970 maps.
>
> In areas covered by ED's for 1980, collection considerations dictate ED size and design, and historical comparability does not normally enter into consideration.

Minor Civil Division (MCD). The primary political and administrative subdivisions of counties. MCD's are most frequently known as townships, but in some States they include towns, magisterial districts, and similar areas. MCD'S are used for census purposes in 30 states (. . .). In the remaining States census county divisions are used in lieu of MCD's.

The Census Bureau has assigned each MCD, in alphabetic sequence within county, an incremental, unique three-digit numeric code. In addition, MCD's in 11 States have a four-digit "MCD sequence number" which allows MCD's to be sorted into alphabetical sequence within a State. MCD's in some States are also assigned a five-digit Federal Information Processing Standards (FIPS) place code which is unique within State. The National Bureau of Standards may expand the coverage of FIPS place codes to include MCD's in the remaining States. If the timing of such an expansion permits, the new codes will also appear with MCD records on tape.

Minor civil division boundaries are represented on all detailed census maps. In addition, MCD outlines appear at a small scale in maps published in PC(1)-A and HC(1)-A State reports. There are approximately 25,000 MCD's defined for the 1980 census.

Statistics for all MCD'S appear in STF'S 1A, 2B, 3A, and (under tentative plans) 4B, and in PC(1)-A and -B, and HC(1)-A reports. In 20 States (. . .), most MCD'S serve as functioning general-purpose governments, and these active MCD's are included in PHC(3) Summary Statistics for Governmental Units. All MCD's in these States are included in PHC(1) Block Statistics reports and STF 1B.

Finally, in 11 States (all 9 States in the northeast region, plus Michigan and Wisconsin), MCD data are published in a manner parallel to that of places of the same size in tables of PC(1)-B and -C, and HC(1)-A and -B. (. . .)

> Historical comparability: Census county divisions were used in North Dakota in 1970, but for 1980 that State returned to the use of its townships.
>
> A number of minor civil divisions in other States have changed boundaries. Some of these have been as a result of municipal annexations in several States. There are six States where MCD boundaries have changed substantially: Arkansas, Virginia, Louisiana, Mississippi, West Virginia, and Maryland. A new set of subcounty areas, termed "census subareas," has been developed for Alaska.

Place. A concentration of population which may or may not have legally prescribed limits, powers, or functions. Most of the places identified in the 1980 census are incorporated as cities, towns, villages, or boroughs. In addition, a number of census designated places (called "unincorporated places" in earlier censuses) are delineated for 1980 census tabulations. There are about 23,000 places recorded in the 1980 census.

Incorporated place. A political unit incorporated as a city, borough (excluding Alaska and New York), village or town (excluding the New England States, New York and Wisconsin). Most incorporated places are subdivisions of the MCD or CCD in which they are located; for example, a village located within and legally part of a township. Some incorporated places are independent of surrounding townships or towns and therefore are also treated as MCD's. Finally, almost 4,000 incorporated places cross MCD and/or county lines. No incorporated places cross State lines since they are chartered under the laws of a State.

There are about 20,000 incorporated places for the 1980 census.

Census designated place (CDP). A densely settled population center without legally defined corporate limits or corporate powers or functions. Each has a definite residential nucleus with a dense, city-type street pattern, and ideally should have an overall population density of a least 1,000 persons per square mile. In addition, a CDP is a community that can be identified locally by place name. Boundaries of CDP's are drawn by the Census Bureau, in

cooperation with State and local agencies, to include, insofar as possible, all the closely settled areas. In the 1980 census, statistics are tabulated for each CDP with 5,000 inhabitants or more if located in an urbanized area with a central city of 50,000 or more and for each CDP of 1,000 or more if in an urbanized area with a central city of less than 50,000. Outside of urbanized areas, statistics are tabulated in 48 States for CDP's of 1,000 or more, in Hawaii for CDP's of 300 or more, and in Alaska for CDP's of 25 or more.

There are approximately 3,000 CDP's.

Incorporated place and CDP boundaries are shown on all detailed census maps. MMS/VMS maps show the boundaries of places in or near urbanized areas, and place maps are available for all places outside MMS/VMS coverage. In tracted areas, boundaries of places with 10,000 or more inhabitants are shown on tract outline maps, which are at a smaller scale than MMS/VMS maps. County subdivision maps, at still smaller scale, show boundaries for places with 2,500 or more inhabitants and pinpoint the location of smaller places.

A 4-digit numeric code is assigned by the Census Bureau to each place in alphabetic sequence within State. In addition, a 5-digit Federal Information Processing Standards (FIPS) place code, unique within State, has been assigned by the National Bureau of Standards for each place. Both codes will appear on computerized records for places, but the 4-digit census code will be used in structuring the files (i.e., in determining the sequence of place records). Separate "place description" codes will also generally accompany place records. These codes indicate whether or not a place is incorporated, as well as represent certain other information about places.

All places are summarized in STF's 1A and 3A and PC(1)-A reports. Places with 1,000 or more inhabitants are summarized in STF 2B, and PC(1)-B and HC(1)-B reports. Places with 2,500 or more are summarized in STF 4B, and PC(1)-C and HC(1)-B reports. Incorporated places only are shown in PHC(3) reports. In PHC(2) Census Tracts reports and STF's 2A and 4A, summaries are presented only for places with 10,000 or more inhabitants located in SMSA's or other tracted areas.

The files and reports which sequence geographic units in hierarchical fashion must account for the fact that places may cross the boundaries of counties, minor

civil divisions and census county divisions. Such reports and tapes, therefore, provide summaries for the various parts of places created when places are split by the boundaries of higher-level areas recognized in the hierarchy. Specifically, place parts within county and MCD and CCD are presented in STF 1A and STF 3A, and PC(1)-A. Place parts within county and MCD are presented for 20 States only in STF 1B and PHC(1) Block Statistics reports, although the PHC(1) reports exclude any place which does not have block statistics. In the remaining 30 States, STF 1B and PHC(1) reports subdivide places when split by county boundaries, but do not observe MCD or CCD boundaries.

Historical comparability: Nearly 65 percent of all incorporated municipalities annexed territory between 1970 and 1977, and this proportion increased further by January 1, 1980, which is the reference date for boundaries in the 1980 census. In the 1970 census, ED boundaries were drawn so as to allow a user to aggregate 1970 data for each city of 2,000 or more inhabitants according to 1960 boundaries. There will not be a corresponding capability in the 1980 census. Instead, a special report on annexations during the 1970's may be prepared if funds are available. Such a report may cover central cities, other than places of 50,000 or more population, and any smaller places which annexed areas with 1,000 or more inhabitants as of 1970. The special report would compare the 1970 and 1980 population of each covered city as defined in 1980 (forward comparability), and

would make a similar comparison for each city as defined in 1970 (backward comparability). This would be the first census report to provide forward comparability.

In the 1970 and earlier censuses, census designated places were referred to as "unincorporated places." The name was changed to make it more explicit that such places are defined for census purposes, and to avoid confusion in States where many "unincorporated places" are parts of incorporated towns or townships. Many census designated places have been redefined since 1970.

Region, (Census). Census regions are large groups of States which are first-order subdivisions of the United States for census purposes. The four regions [are] Northeast, North Central, South, and West . . . . Census regions have no relationship to the 10 Standard Federal Administrative Regions. Regions are identified by a one digit code. Regions are summarized in U.S. Summary reports in almost every publication series, and in STF's 1C, 2C, 3C, and 4C.

Standard Consolidated Statistical Area (SCSA). A large concentration of metropolitan population composed of two or more contiguous SMSA's which meet certain criteria of population size, urban character, social and economic

integration, and/or contiguity of urbanized areas. Each SCSA must include at least one SMSA with a population of one million or more. Thirteen SCSA's are defined for the 1980 census by the Office of Federal Statistical Policy and Standards according to criteria published by that office in Standard Metropolitan Statistical Areas: 1975.

SCSA's are identified by a two-digit numeric code. Summaries for SCSA's appear in all reports, except PHC(1) and PHC(2), and in STF's 1C, 2C, 3C, and 4C. Summaries are generally provided for SCSA totals and for within-State parts of SCSA's.

Historical comparability: The 13 SCSA's were first created in 1976. For the 1960 and 1970 censuses the Census Bureau recognized two "Standard Consolidated Areas" (SCA's), metropolitan complexes around New York and Chicago.

In 1982 or 1983, the SCSA concept will be replaced by the new Consolidated Metropolitan Statistical Area (CMSA) concept, with somewhat more liberal criteria, as spelled out in the Federal Register, Jannuary 3, 1980. These changes will not affect publication of 1980 census data for SCSA's.

Standard Metropolitan Statistical Area (SMSA). A large population nucleus and nearby communities which have a high degree of economic and social integration with that

nucleus. Generally, each SMSA consists of one or more entire counties that meet specified standards pertaining to population, commuting ties, and metropolitan character. In New England, towns and cities, rather than counties, are used as the basic geographic units for defining SMSA's. In Alaska, boroughs and census areas are used for defining SMSA's. SMSA's are designated by the Office of Federal Statistical Policy and Standards of the Department of Commerce.

SMSA's to be observed in the 1980 census are of two types: (1) those defined before January 1, 1980, 288 in all (including 4 in Puerto Rico); and (2) those to be established in 1981 as a result of 1980 census population counts. In order for a new SMSA to be recognized following the 1980 census, an area must have either:

1. A city with a population of a least 50,000 within its corporate limits, or

2. A Census Bureau defined urbanized area (which must have at least 50,000 population) and a total SMSA population of at least 100,000.

Each SMSA includes not only a city and its urbanized area, but also the remainder of the county or counties in which they are located and such additional outlying counties as meet specified criteria relating to metropolitan character and level of commuting of workers into the central city or counties. Specific criteria governing the definition of SMSA's recognized before 1980 are published in Standard Metropolitan Statistical Areas: 1975, issued by the Office of Federal Statistical Policy and Standards.

With two exceptions, each SMSA has one or more central cities, up to a maximum of three, and the names of these cities comprise the title of the SMSA. The Nassau-Suffolk, NY SMSA has no central cities; the Northeast Pennsylvania SMSA has three: Scranton, Wilkes-Barre, and Hazleton.

SMSA's are identified by a Federal Information Processing Standards (FIPS) 4-digit numeric code, which follows the alphabetic sequence of the SMSA name. SMSA's are outlined on small scale maps in several 1980 report series. SMSA data appear in most 1980 census publications and summary tape files. Many SMSA's cross State boundaries, and several reports provide summaries for the within-State parts of multi-State SMSA's as well as SMSA

totals. Summary tape files present data only for within-State parts of SMSA's, except for the "national" files: STF's 1C, 2C, 3C, and 4C.

Historical comparability: Since the 1970 census, when 247 SMSA's were recognized in tabulations (including 4 in Puerto Rico), a number of new SMSA's have been defined. Of the 247 1970 SMSA's, 101 were redefined in 1973, based on 1970 census results, most by the addition of one or more counties (or towns and cities in New England). In addition, one SMSA was redefined by the addition of one area and the deletion of another (Wichita Falls, Texas), one was subdivided (Nassau-Suffolk SMSA was created from a part of the New York SMSA), four pairs of SMSA's were combined into single SMSA's (for example, Dallas-Fort Worth, Texas), and four SMSA's lost area that was added to other SMSA's. In addition, the names of several SMSA's were changed in 1973, one in such a way that the SMSA code also changed (San Bernardino-Riverside-Ontario to Riverside-San Bernardino-Ontario, California).

Since SMSA's are always defined in terms of whole counties (towns or cities in New England) and extensive data have been and will be available on tape for these areas, forward and backward comparability can usually be derived by the user.

In 1982 or 1983, SMSA boundaries will be re-evaluated using 1980 census data on commuting, population density, type of residence and population growth, according to new criteria spelled out in the Federal Register, January 3, 1980 (Vol. 45, No. 2, Pt. VI). At that time new outlying counties may be added or existing ones dropped, some area titles may change, many new central cities will be designated, and some areas may be consolidated. Further, the term "standard metropolitan statistical area" will be shortened to "metropolitan statistical area" (MSA). These changes will not affect publication of 1980 census data for SMSA's.

State. The major political units of the U.S. The District of Columbia is treated as a State-equivalent in all 1980 census data series.

States are identified by a two-digit Federal Information Processing Standards (FIPS) code which follows the alphabetic sequence of State names, and [by] a two-digit Census geographic State code, the first digit of which identifies the census division of which the State is a part. The Federal Information Processing Standards (FIPS) State code is used for sequencing in most reports and in STF's which present data for all States.

Historical comparability: There have been no significant changes to State boundaries in the last several decades.

Town/Township. (See Minor Civil Division)

Tract. Relatively small areas with generally stable boundaries into which metropolitan and certain other areas are divided for the purpose of providing statistics for small areas. When tracts are established, they are designed to be relatively homogeneous areas with respect to population characteristics, economic status, and living

conditions. The typical tract contains between 2,500 and 8,000 residents. All SMSA's recognized before the 1980 census are completely tracted. In addition, nearly 4,000 census tracts have been established in 252 counties outside those SMSA's (although some of these areas are likely to become SMSA's as a result of the census). In fact, 5 States have been entirely tracted: Rhode Island, Connecticut, New Jersey, Delaware, and Hawaii. In all, there are over 42,000 census tracts for the 1980 census.

Tract boundaries are established cooperatively by local Census Statistical Areas Committees and the Census Bureau in accordance with guidelines that impose limitations on population size and specify the need for visible boundaries. Geographic shape and areal size of tracts are of relatively minor importance. Tract boundaries are established with the intention of being maintained over a long time so that statistical comparisons can be made from census to census. Census tracts observe county lines and are defined to cover all of the territory within each tracted county.

Census tracts are identified by a 4-digit basic code and a 2-digit suffix, e.g., 6059.02. Many census tracts do not have a suffix. In such cases, maps show just the 4-digit code; tapes give the 4-digit code followed by two

zeros. Tract numbers are always unique within a county, and, except for New York, are also unique within an SMSA. All valid census tract numbers are in the range 0001 to 9899.99; a number between 9901.00 and 9989.99 denotes a block numbering area.

Census tract boundaries are shown on all detailed census maps. In addition, census tract outline maps are being created for each SMSA and each tracted county outside SMSA's and will be published in PHC(2) Census Tracts reports. Tract outline maps show streets and physical features which serve as census tract boundaries. In addition, tract outline maps show the boundaries for places with 10,000 or more inhabitants and for counties.

Census tract data are presented in STF's 1A, 1B, 2A, 3A, and 4A, and in PHC(2) Census Tracts reports. In STF 1A and STF 3A, tract data are presented in hierarchical sequence within place within MCD or CCD. Since a tract which crosses place, MCD, or CCD boundaries is split, the tape files will have summaries for each of its parts. To get data for the whole tract, it will be necessary to add up the components. In STF 1B the situation is similar except that MCD boundaries are observed in only 20 States. (. . .) MCD boundaries in the other 10 States with MCD's and CCD boundaries in the remaining 20 States are ignored.

In the major summaries for census tracts, those in STF 2A and STF 4A and in PHC(2) Census Tracts reports, tract summaries observe the boundaries of places of 10,000 or more. There are also separate summaries which provide totals for split tracts.

Historical comparability: Census tracts are defined with an overall goal of census-to-census comparability. Some 1970 tracts have been subdivided due to increased population, but the new tracts can be recombined by the user for comparison with 1970 tracts. This affects about eight percent of all 1970 tracts. Other changes have included combinations of two or more small 1970 tracts (less than one percent of all 1970 tracts) and adjustments to tract boundaries where old boundary features have disappeared or new boundaries (e.g., freeways) have come into being. Only in a few areas did local Census Statistical Area Committees undertake extensive redefinition of census tracts.

Both the number of tracted counties and the number of census tracts increased by over 20 percent between 1970 and 1980. The reporting of data for split tracts is also increasing. Where 1970 Census Tracts reports gave data for tract parts created when tracts were split by the boundaries of only those places with 25,000 or more population, 1980 reports will observe places as small as 10,000. 1980 STF's 2 and 4 summarize data for split tracts as well as whole tracts, whereas their 1970 counterparts did not provide separate summaries for split tracts.

Urban and rural area (population). Urban and rural are type-of-area concepts rather than specific areas outlined

on maps. As defined by the Census Bureau, the urban population comprises all persons living:

1. in urbanized areas (defined below) and

2. in places of 2,500 or more inhabitants outside urbanized areas.

The rural population consists of everyone else. Therefore, a rural classification need not imply farm residence or a sparsely settled area, since a small city or town is rural as long as it is outside an urbanized area and has fewer than 2,500 inhabitants. (. . .)

The terms urban and rural are independent of metropolitan and nonmetropolitan designations: both urban and rural areas occur inside and outside SMSA's.

> Historical comparability: Except for the minor liberalization of urbanized area criteria discussed below, urban and rural definitions have been consistent since 1950. Within small counties, measurements of urban and rural populations over time may be disproportionately affected by the increase or decrease of a place's population across the 2,500 population threshold, e.g., the increase of 1 person to a place of 2,499 results in an increase of 2,500 to the area's urban population.

Urbanized area (UA). A population concentration generally consisting of a central city of 50,000 inhabitants or more and the surrounding, closely settled, contiguous territory (suburbs). In addition a population concentration may qualify as an urbanized area if it has a central city of between 25,000 and 50,000, and which, when incorporated places and census designated places adjacent to the city limits are included, has a total population of at least 50,000. (For 1980, the minimum size requirement for central cities may be dropped. This would mean the only requirement for the establishment of an urbanized area would be a densely settled population concentration with a minimum total population of 50,000.)

The urbanized area criteria define a boundary based primarily on a population density of at least 1,000 persons per square mile, but also include some less densely settled areas within corporate limits, and such areas as industrial parks and railroad yards, if they are adjacent to dense urban development. The density level of 1,000 persons per square mile corresponds approximately to the continuously built-up area around a city. The "urban fringe" is that part of the urbanized area outside of a central city. (. . .)

Typically, an entire urbanized area is included within an SMSA. The SMSA is usually much larger in terms of territory covered and includes territory where the population density is less than 1,000. There occasionally are more than one UA within an SMSA. In some cases a small part of a UA may extend beyond an SMSA boundary, and possibly, into an adjacent SMSA. A few 1980 UA's will be defined in areas which do not meet the 100,000 total population criterion for SMSA designation. UA's may cross State boundaries. In a few cases a UA may not include all of an "extended" central city which is determined to have a significant amount of a rural territory.

UA's are identified by 4-digit codes, which follow the alphabetic sequence of UA names. Their boundaries will be shown on final Metropolitan Map Series and Vicinity Map Series maps and, at much smaller scale, on UA maps in PC(1)-A and HC(1)-A reports.

Historical comparability: Because UA's are defined on the basis of population distribution at the time of a decennial census, their boundaries tend to change from census to census.

The criteria have been fairly constant since 1950, although in each decade some new refinements have been added. For the 1970 census, in which 252 UA's were recognized, it was necessary for the central city to have a population of 50,000 or more, or for there to be "twin cities" with a combined population of

50,000 and with the small city having at least 15,000. In 1974 the criteria were liberalized to allow UA recognition to certain cities between 25,000 and 50,000, and this resulted in 27 new urbanized areas. As noted earlier, another revision of the urbanized area criteria is now under consideration.

A P P E N D I X C

A SYSTEM ARCHETYPE

We have described the makeup of a geographic information system (GIS) which is based on our GERMS data modeling formalism. In this appendix we detail the structure of an actual system implementation. The database management system which forms the core of this system is SIR-2 (Scientific Information Retrieval, version 2). We will hereunder refer to such as SIR.

The SIR database management system (DBMS) includes an extensive high level programming language through which programmed database operations can be specified. We use this language to form a "GERMS front-end" to the database. In this way, all user-system communications are carried out in terms of the GERMS conceptual schema. This is despite the fact that the basic SIR database system is not oriented towards the support of semantic data models.

An Introduction to SIR and the SIR Language

In this section we describe those features of the SIR system which are fundamental to an understanding of our GIS implementation. The reader who is familiar with SIR may omit this section. We caution the reader that this familiarity must be with SIR version 2, as this version differs substantially from its predecessor.

A SIR structure utilizes both cases and record types. A case is a collection of records of one or more record types. Usually, a case relates logically to a single entity or subject. Each case is identified by a unique case name.

A SIR record type is a collection of records which have a common format and attributes. Each record type does not exist within its own file in SIR. Every record type is identified by a record type number and by a (optional) record type name. Members of a record type are record instances. The membership is nonoverlapping, thus an instance is not shared by types.

A record type can be (and in most applications is) shared by all cases in the database. Each record instance belongs to exactly one case, however. When designing a SIR database we have the option of assigning all instances

of a given record type to a single case. Here, we can think of the record type as "belonging" to the case. We will make use of this technique often in our system.

A record instance is identified by its unique record key value. Part of this value is its case name and its type identifier. In addition to these identifiers the record key may consist of the concatination of a number of record variables. Each such record variable is called a "sort id."

When different record types make use of the same sort ids, the records form a hierarchy. For example, if record type 1 has the single sort id "KEY1," and record type 2 has the sort id pair "KEY1, KEY2," then the type 2 records are children of the type 1 records. "Ownership" of the children record instances occurs when type 1 and type 2 records hold the same value on variable KEY1.

SIR is designed to accommodate the processing of hierarchical structures. Also, the processing of network structures can be emulated with the clever use of pointer variables. Given the key value (including case name and record type identifier) of any database record instance, the record can be accessed by the system. By embedding the key value of a record instance within the data of a different record, then, logically related records can be "pointed to." Network structures can be designed thusly.

Unlike hierarchical record relationships, the use and maintenance of such "network emulating" pointers is the responsibility of the database technician or user. Consequently, we call SIR an "extended hierarchical" DBMS; the ability to "jump" between hierarchical structures can be programmed (note that any network structure can be viewed as a set of trees).

The SIR language is used to manipulate and retrieve data by making use of the relationships among records. It can therefore be thought of as being the data manipulation language of the DBMS. Programs written in this language can serve as "front-ends" to the database. The flexibility of the SIR language allows for the creation of data retrieval procedures which meet the specific needs of the user community.

The language incorporates features that are common to most high level symbolic programming languages (i.e. looping, branching, conditional statements, callable procedures, etc). Obviously, these features are provided in addition to record structure processing capabilities.

Within a SIR program the following statement is used to specify the case instance which is to be the focus of the processing:

CASE IS caseid

where "caseid" is the unique case name of the instance.

Once the case is defined, record instances can be specified in a number of ways. The statement

RECORD IS rtype, (keyval)

identifies the single instance of record type "rtype" which holds the key value "keyval" (note that keyval may be the concatenation of a set of values).

Within the specified case, the set of all records of type "rtype" can be processed by stating

PROCESS REC rtype

The set of siblings (of a given type) of a given parent instance can be processed via

PROCESS REC rtype, WITH (sortidlist)

Where "sortidlist" is the value of the key of the parent

record.

Consider, for example, that two records are related hierarchically such that the sort id of the superordinate record is "KEY1," and the sort id of the subordinate record type (rtype) is "KEY1, KEY2." In order to process the set of siblings which belong to the instance of the first record type which has the current value of KEY1 as its key, we need only state the following:

PROCESS REC rtype WITH (KEY1)

Alternatively, if we wish to specify the single instance of the latter record type which has the concatenation of the current values of KEY1, KEY2 as its key, we should state

RECORD IS rtype, (KEY1, KEY2)

Each of the above record (or case) definition formats is called a "block header;" it indicates the start of the appropriate record (case) "processing block." The terminus of a processing block is indicated by the presence of a matching END statement (e.g. END CASE IS, END PROCESS REC). All program statements and variable references which lie within a block refer to the records (case) defined by the block header. With proper nesting of processing blocks, complex record structures can be processed. Statement indentation will be used to clarify the nesting of processing blocks within SIR routines.

Both the clarity and the flexibility of SIR programs can be increased through the use of procedures and substitution parameters. In SIR, a procedure is a named set of statements which may be "called" from within a SIR program. When a CALL statement is encountered during the execution of a program, the statements which form the procedure are directly inserted in place of the CALL statement. In terms of the calling program, then, program execution is carried out as if the statements of the procedure exist as a part of the program.

A CALL statement can include an optional substitution parameter list. The elements of such a list are called "positional parameters" because their names are

assigned according to their position in the list. Specifically, a parameter's name is its sequence number.

Consider, for example, the following CALL statement:

```
CALL procname (param1, param2,...paramN)
```

Here, each of param1, param2,...paramN represents some character string which is the parameter's value. When this CALL statement is encountered, the string value "param1" is assigned parameter name "1." Similarly, "param2" is assigned parameter name "2," and so on up to and including "paramN."

Within the body of the procedure a positional parameter name enclosed in angle brackets denotes a reference to that parameter. Before the statements of the procedure are inserted into the CALLing program, each parameter reference is replaced with the appropriate parameter value.

Let us assume that the statement

```
CALL PROC1 (POP GT 10000, NEXT RECORD)
```

is encountered in the execution of a SIR program. As described above, "POP GT 10000" is assigned parameter name "1," while "NEXT RECORD" is assigned parameter name "2." Let us further assume that the procedure PROC1 contains the following statement:

IF (<1>) <2>

Now, each parameter reference in a statement of a procedure is replaced with its parameter value prior to the insertion of the statement into the CALLing routine. Consequently, the inserted statement -- that which is actually executed -- is

IF (POP GT 10000) NEXT RECORD

Another form of substitution parameter is the "global parameter." Global parameter values, which are character strings, are assigned alphanumeric character names by means of the GLOBAL statement. Subsequent to the

assignment of a global parameter name, each parameter reference (the parameter name enclosed in angle brackets) within the body of a program statement is replaced by the appropriate value. Obviously, this replacement is made prior to the execution of the statement.

For example, consider the GLOBAL statement

```
GLOBAL CONDITIN = POP GT 10000/
       TASK = NEXT RECORD
```

Here, the parameter (string) value "POP GT 10000" is equated to parameter name "CONDITIN," while the value "NEXT RECORD" is equated to "TASK." Following the execution of this GLOBAL statement, each parameter reference is replaced by the appropriate parameter value. Thus the program statement

```
IF ( <CONDITIN> ) <TASK>
```

would be interpreted as

```
IF (POP GT 10000) NEXT RECORD
```

The SIR routines discussed below rely heavily on the use of substitution parameters. The reader who wishes more detail about these or other aspects of SIR is referred to [ROBI80].

System Design

The design of our GERMS implementation matches closely the configuration detailed in chapters 11, 12, and 13. As described earlier, the design comprises two major divisions: the system internal form, or logical database; and the physical database. The physical database holds the actual data records, directories, and indexes. The system internal form holds the system's "knowledge" of the logical data structure, and "links" this logical structure to the objects of the physical database.

As our example data set we will use a portion of the aforementioned census file PL-94. The relevant GERMS conceptual schema graph is shown in figure C.1. The logical structure of this data set is not complex, so an extensive description is not required here. The schema does not happen to contain any "IS-A-related" entity types, but the proper treatment of such objects will become obvious from the subsequent discussion.

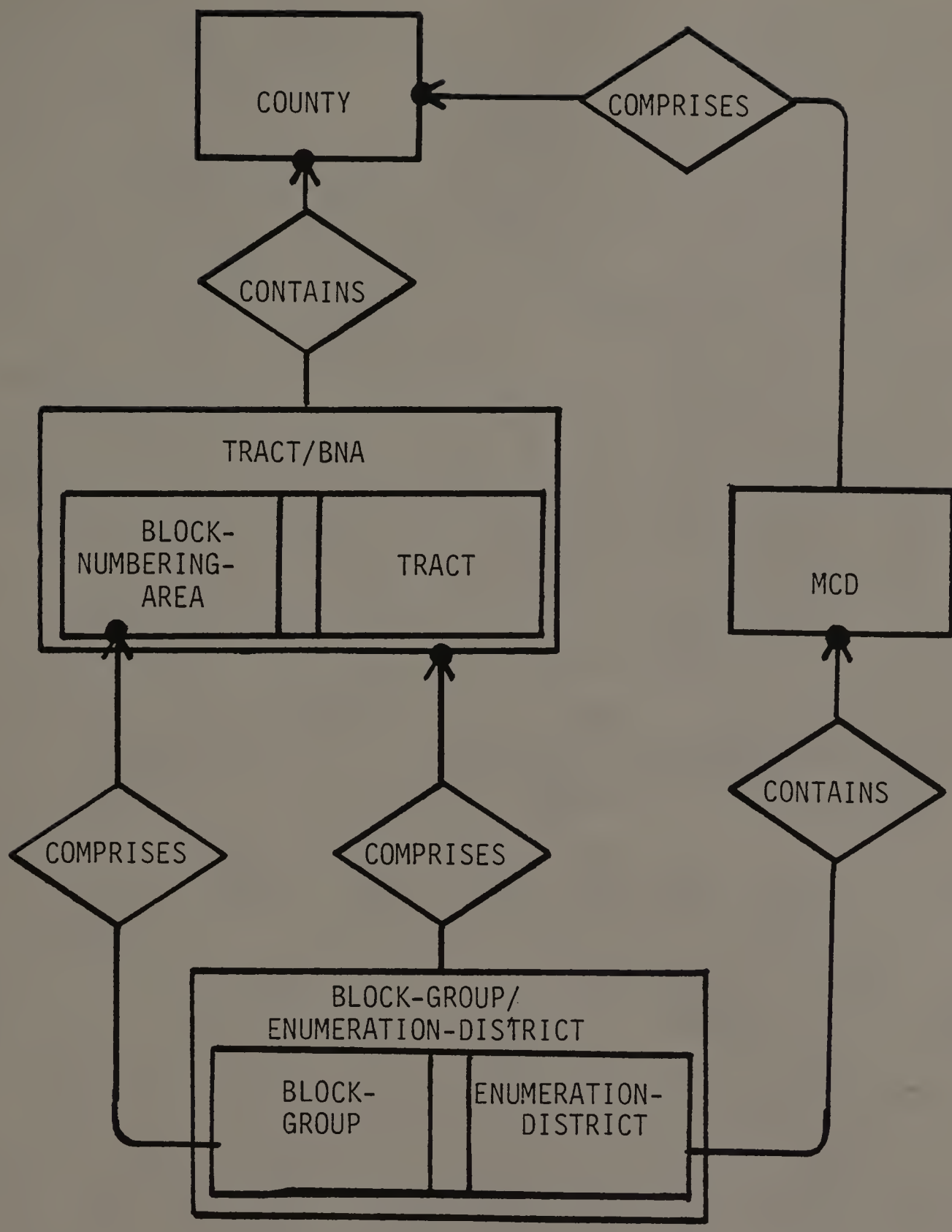


FIGURE C.1

THE DATABASE CONCEPTUAL SCHEMA

The physical database. In our SIR configuration, the physical database utilizes four case instances and thirteen record types. Of the thirteen record types, four are used to hold actual census summary data. We will call these "data record types." The nine remaining record types are used to serve as directories and indexes. These will be referred to as "pointer record types." More specifically, they may be "directory types," or "index types."

The four cases are named 'ENTITY2', 'ENTITY4', 'ENTITY6', and 'ENTITY8' (the use of even integers here occurs by coincidence). Note that each of these case names is a string value. Single quotes are used here and below to distinguish string values from variable names.

Each case in the physical database relates to a specific entity type of the logical structure. Specifically, the cases relate to the BLOCK-GROUP/ENUMERATION-DISTRICT, COUNTY, MCD, and TRACT/BNA entity types, respectively. Each case has exclusive ownership of a census data bearing (vis. pointer bearing) record type. The case ownership of record type numbers (and type names) is as follows:

1. Record type 2 (BLOCK-GROUP/ENUMERATION-DISTRICT) belongs to case 'ENTITY2'

2. Record type 4 (COUNTY) belongs to case 'ENTITY4'

3. Record type 6 (MCD) belongs to case 'ENTITY6'

4. Record type 8 (TRACT/BNA) belongs to case 'ENTITY8'

Notice that the case name integer (i.e. 'ENTITY*n*') is the same as the record type number of its data record type.

Figure C.2 shows the portion of the physical database discussed to this point. Each *ellipse* represents a *case instance.* The case name (and associated entity type) is contained in the ellipse. Each *rectangle* represents a specific *data record type.* The record type number (circled) and record type name is shown. An undirected arc represents exclusive ownership of a record type by a case instance.

The reader should note that the physical database uses only *four data record types,* while the conceptual schema references *eight logical entity types.* This physical design is possible because the schema employs

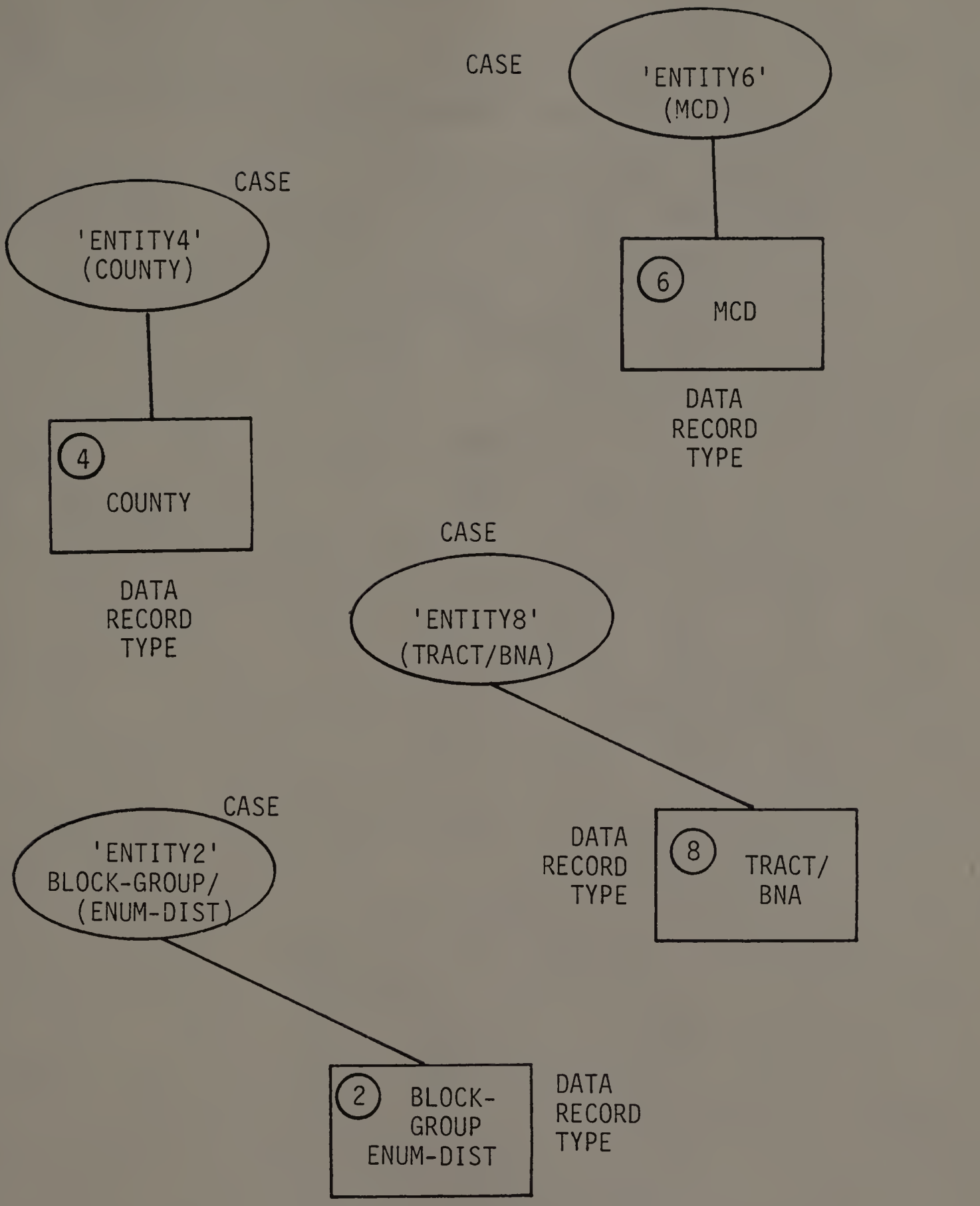


FIGURE C.2

THE PHYSICAL DATABASE: CASES AND RECORD TYPES

"blatant logical redundancy;" more than one logical object refers to a single physical object. This is true in the case of the BLOCK-GROUP/ENUMERATION-DISTRICT logical entity type and in the case of the TRACT/BNA entity type. As described in chapter 11, the instances of the "lower level" constituents of these "higher level" entity types can be represented by record indexes. This technique allows for the avoidance of physical redundancy in the data representation.

Record type 1 (BLOCK-GROUP), type 3 (BLOCK-NUMBERING-AREA), type 5 (ENUMERATION-DISTRICT), and type 7 (TRACT) serve as indexes for the "lower level" entity types. The record format for each of these pointer record types is identical. Each such record holds a single value: the key of the area which it represents. Obviously, this is also the key of the record. Each "lower level" logical entity type instance is represented by exacly one pointer record instance, so each of these pointer record types simply serves as a list of key values which are of the respective logical entity type.

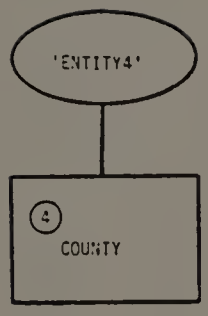
The value of the key of a "lower level" area instance is the same as that of its "higher level" counterpart (they are the same physical object). As such, the respective pointer record and data record instances

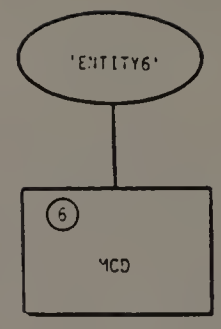
have the same record keys. Every index record therefore "points" to its appropriate data record. In this way the database configuration supports the different views (higher level/lower level) of the common non-redundant data. "IS-A-related" entities can be dealt with in a similar manner.

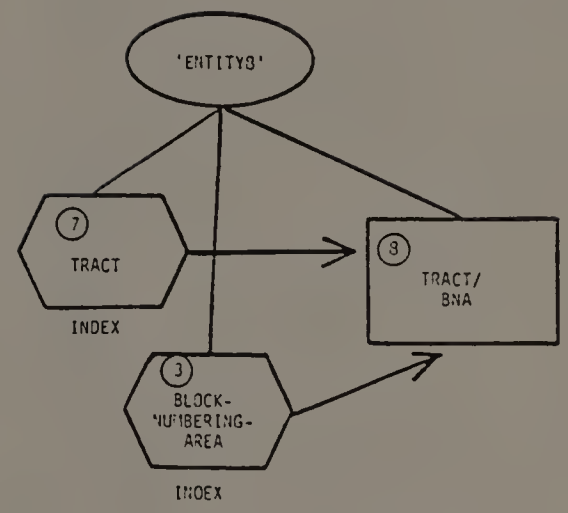
Figure C.3 shows the physical database as described so far. Each hexagon (flowcharting preparation symbol) depicts a pointer record type which is used as an index. The directed arcs are used to magnify the "pointer aspect" of the indexes. Note that when a record type is used as an index, the "target" record type of the index belongs to the same case instance.

Up to this point we have discussed eight record types (numbered 1 through 8) of the physical database. The remaining five record types (9 through 13) represent the relationships of the conceptual schema; they are used to emulate the inverted list directory structure described in chapter 11. Inverted lists must be emulated because SIR does not support this structure. We will see that the technique used here serves the same purpose.

Recall from the chapter 6 description that the purpose of a relationship directory is to "link" the superordinate (in a geographic relationship) "entity"







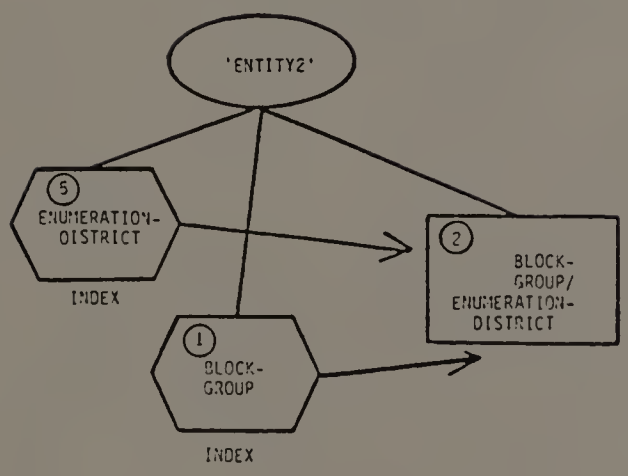


FIGURE C.3

INDEX RECORDS

instance with its set of subordinate "entity" instances. Each relationship instance represents a "one-to-many" mapping from superordinate to subordinate entity type. The data structure used to represent a relationship instance must support "downward" processing of the involved objects.

In our SIR configuration each relationship directory is represented by its own record type. Each directory record instance relates to a single instance of the subordinate entity type. Thus, a superordinate record "points to" a number of directory records (notice that, structurally, this is quite different from an inverted list).

The format of a directory record depends on whether or not the subordinate type represents a "special case" entity, that is, one which is accessed through special indexes as described above. In the event that the subordinate type does not represent a special case entity, a directory record holds two values. The record format is as follows:

KEY, PTR

Here and below, record key values are enclosed in "$" characters.

The first record value, KEY, is the key value of the superordinate instance which points to the directory record. The remaining record value, PTR (PoinTeR), is the key of the subordinate instance which is pointed to by the directory record. Note that the key value of the directory record, which must be unique within the record type, is formed by the concatination of these two values.

In the other situation, when the subordinate does represent a special case, a directory record holds three values. The record format is

$KEY, TYPE(string), PTR$

The meaning of KEY and PTR are as before. The string value TYPE holds the "least abstract" entity type name of the object pointed to. For example, if the subordinate entity type is BLOCK-GROUP/ENUMERATION-DISTRICT, then each entity instance is either a BLOCK-GROUP entity or an enumeration-district entity. The value of TYPE indicates which. "IS-A-related" entities

are treated similarly. Note that the key value of the record type is formed as the concatination of these three record values in this latter case.

Now, if we wish to access all subordinate BLOCK-GROUP/ENUMERATION-DISTRICT instances that are related to a given superordinate record with key value KEY, we can state

```
PROCESS REC rtype, WITH (KEY)
  COMPUTE NEWKEY = PTR
  RECORD IS 2, NEWKEY
      .
      .
  END RECORD IS
END PROCESS REC
```

where "rtype" is the directory record number. If, on the other hand, we wish to access all subordinate ENUMERATION-DISTRICT instances that are related to the superordinate record with key value KEY, we need only replace the above first statement with

```
PROCESS REC rtype, WITH (KEY, 'ENUMERATION-DISTRICT')
```

Obviously, the second sort id can be specified as a string variable name.

Note that each of these statement forms involves the same record type since we are dealing with logical but not physical redundancy. The latter statement places a greater restriction on the selection of records. Thus the records referenced in this latter format are a subset of those referenced in the former.

Because of the flexibility of this relationship directory technique, a number of relationships can be represented by a single directory. This obviously reduces greatly the storage space required for the directories. All relationships of our example conceptual schema, including the implied relationships, are represented by only five directories (record types). These five directories depict fifteen different geographic relationships.

Figure C.4 shows the entire physical database configuration. Each triangle depicts a directory pointer record type. The meanings of the remaining symbols are as before.

Some points deserve noting here. First of all, the directory record type is "owned" (hierarchically) by the superordinate data record type which points toward the

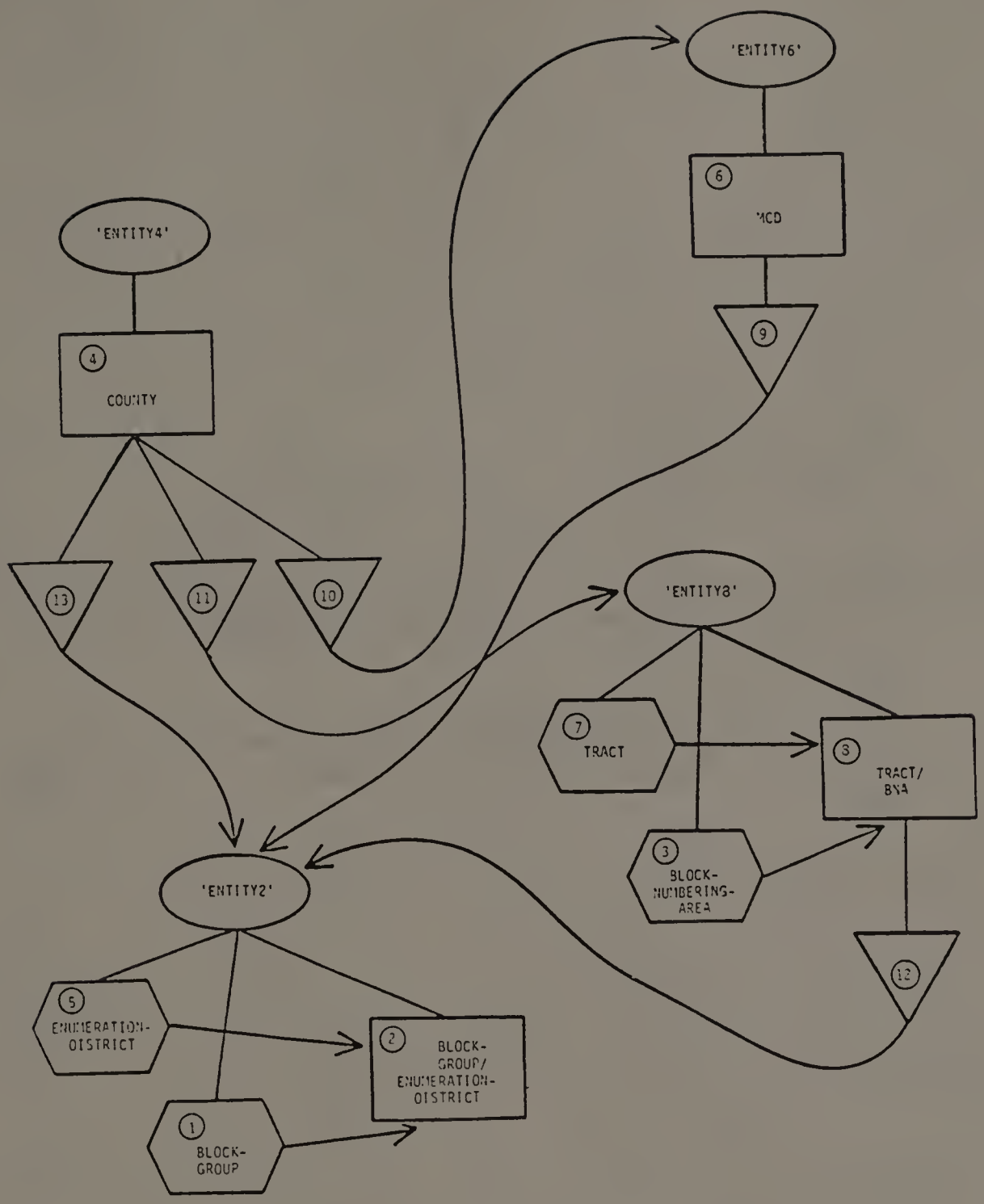


FIGURE C.4

THE FULL PHYSICAL DATABASE

directory. Second of all, each directory points outside its case instance. Finally, as described previously, the same directory is used regardless of whether or not the superordinate or the subordinate (or both) is a "special case" entity type; any case instance pair is joined by at most one directory.

The system internal form. Under the methodology put forth in chapter 12, the system internal form is made up of three distinct parts. Specifically, these are the symbol table, the schema matrix, and the pointer matrix. In our SIR configuration each of these parts is represented by a separate record type. These record types are named SYMBOL, SCHEMA, and POINTER, respectively.

Every SIR record instance must be linked to a single case instance. In the design described here all instances of these three record types belong to the single case 'IFORM' (Internal FORM). This is illustrated in figure C.5. Each of the three record types is assigned a unique record number, but these numbers are not pertinent to our discussion.

There is eactly one record instance of type SYMBOL for each entity type of the conceptual schema. Thus, in our example system, there are eight SYMBOL records.

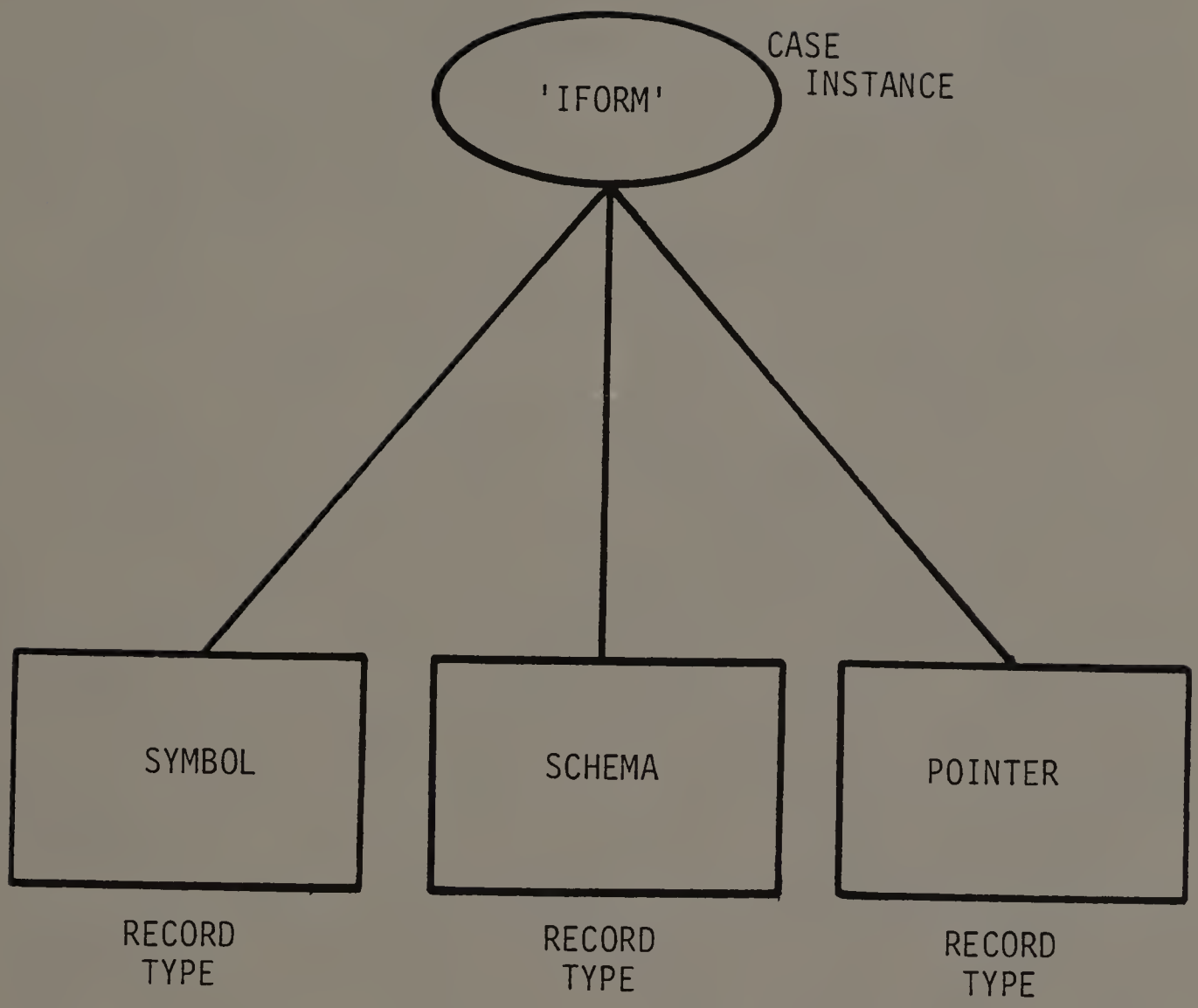


FIGURE C.5

THE SYSTEM INTERNAL FORM

A SYMBOL record holds three values. The format is

$ENAME(string)$, SEQNO, RECNO

ENAME (Entity NAME) is a string value which is the standard GERMS name of the entity type to which the record relates. ENAME is the record key so, given an entity type name, the system can go directly to the proper "row" of the "symbol table."

The unique second value, SEQNO (SEQuence Number), is the sequence number of the entity type represented by the record. In the example discussed here the sequence numbers range from 1 through 8. The third value, RECNO (RECord Number), is the record number in which the data for the entity type is stored in the physical database. These values need not be unique within the record type since our physical storage methodology does not employ physical redundancy in representing logical redundancy. The range of possible RECNO values is the same as that of SEQNO values.

Figure C.6 shows the eight SYMBOL record type instances which relate to our example database. Again, "$" characters enclose the record key. When considering

this figure in conjunction with the previously described physical database configuration, the rationale behind our choice of physical database record type numbers becomes obvious (refer back to figure C.3).

The value of SEQNO for an entity type denotes the record type which is the "first step" in accessing the data for the entity type. Thus, if the data for an entity type is indexed (e.g. TRACT), then SEQNO "points" to the appropriate index record type (i.e. record type 7). If, on the other hand, the data for an entity type is not indexed (e.g. TRACT/BNA), then SEQNO "points" directly to the data record type (record type 8).

The value of RECNO for an entity type denotes the record type which is the "final step" in accessing the data for the entity type. This is, in all cases, a data (vis. pointer) record type. It should be noted that only in the situation of "special case" entity types are the value of SEQNO and RECNO unequal. Consequently, the logical result of

```
IF (RECNO NE SEQNO)
```

can be used to determine whether an index is to be

$ETYPE$	SEQNO	RECNO
BLOCK-GROUP	1	2
BLOCK-GROUP/ENUMERATION-DISTRICT	2	2
BLOCK-NUMBERING-AREA	3	8
COUNTY	4	4
ENUMERATION-DISTRICT	5	2
MCD	6	6
TRACT	7	8
TRACT/BNA	8	8

FIGURE C.6

SYMBOL RECORD TYPE INSTANCES

processed.

Before a SIR record can be accessed, the single case instance of which the record is a member must be made "current." In our physical database design each case instance holds only one data record type. The index record type(s) which point to the data records, and the directory record types which are pointed to by the data records are members of this same case. The problem is that the system must access the proper case before record processing can begin. This problem is easily solved by making use of information contained in the SYMBOL records.

Recall that if the data relating to an entity type are stored in record type "i," then the name of the case in which the data records are stored is string value 'ENTITYi'. Recall also that, for any entity type, the record type number in which its data are stored is held as the integer value of the variable RECNO of the appropriate SYMBOL record.

When processing a SYMBOL record, then, the appropriate case name can be stored in variable _CASENAME as follows:

```
COMPUTE CASENAME = 'ENTITY' + (FORMAT (RECNO, 1))
```

This FORMAT function specification converts the first function argument to a single character string representation of its integer value. Thus, if RECNO holds the value "i," CASENAME is loaded with string value 'ENTITYi'.

Following this computation, the appropriate case can be made current with the statement

```
CASE IS CASENAME
```

SIR does not support the use of matrixes, but "matrix-type" operations can be simulated using records and variables. This technique will be used to emulate operations on the schema "matrix" and the pointer "matrix" as detailed in chapter 12.

Recall that, when the conceptual schema involves N entity types, the internal form schema matrix is of dimension N x N. In our example here the schema deals with eight distinct logical entity types, so we must represent an 8 x 8 matrix. Each matrix row is depicted by a separate record instance. Therefore, record type SCHEMA holds exactly eight record instances.

Given a row-column index pair (i.e. "i,j" of SCHEMA(i,j)) we must be able to access the appropriate matrix element. With our record structure the first step in this process is to access the single record which corresponds to row i of the matrix. Each matrix row corresponds to a specific logical entity type which, in turn, corresponds (via the symbol table) to a specific sequence number (SEQNO). If this sequence number is defined as the record key, then a matrix record instance can be directly accessed through specification of its entity type number. In order to maintain consistency of the meaning of variable names in our internal form structure, this key will be named SEQNO.

As for matrix columns, the SCHEMA record type defines eight non-key values; one for each logical entity type. Thus, each record instance holds a total of nine (in general N + 1) values. The record format is as follows:

$SEQNO$, REL1, REL2, REL3...REL8

For any SCHEMA record instance with key value i, the position "RELj" (RELationship j) relates to the "i,j" element of the internal form schema matrix [1]. SIR provides a "shorthand" method of specifying a "matrix column." Specifically, in the situation described here, the jth "column" can be specified as

REL1 TO REL8 (j)

For any "i,j" element of the schema matrix, the element's value has to do with the relationship (if any) held between logical entity type i and logical entity type j. To review briefly:

"1" denotes CONTAINS

"2" denotes COMPRISES

"3" denotes IS-A

"4" denotes EITHER-OR

"0" denotes "otherwise involved"

"BLANK" denotes "uninvolved"

These concepts were detailed in chapter 12.

In many situations, for instance in verifying the semantic integrity of a query, our system must insure that a CONTAINS ("1") or a COMPRISES ("2") relationship holds between a given entity type pair. The absence of one of these relationships indicates an error.

Now, if the sequence number value of the superordinate entity type is held in variable ROW, and the sequence number of the subordinate entity type is held in variable COL, then the query integrity can be verified as follows:

```
RECORD IS SCHEMA (ROW)
  COMPUTE RELTYPE = REL1 TO REL8 (COL)
  IFNOT (RELTYPE EQ 1 OR 2) error
END RECORD IS
```

Note that use of this procedure demands that the "current" case instance is 'IFORM'.

Figure C.7 shows the actual SCHEMA record instances which relate to our example. All relationship instances, including implied relationships, are shown here. A dash (-) is used to represent the aforementioned BLANK character. The SEQNO key value is used for the purpose of record access only.

$SEQNO$	REL1	REL2	REL3	REL4	REL5	REL6	REL7	REL8
1	-	0	0	0	0	-	0	0
2	4	-	-	0	4	-	0	0
3	2	-	-	0	-	-	0	0
4	1	1	1	-	1	2	1	1
5	0	0	-	0	-	0	0	0
6	-	-	-	0	1	-	-	-
7	1	1	0	0	1	-	-	0
8	1	2	4	0	1	-	4	-

"-" = BLANK

FIGURE C.7

SCHEMA RECORD INSTANCES

The POINTER record type, which likewise holds eight record instances, is used to support "matrix-type" operations in a similar way. Recall that the "i,j" element of the internal form pointer matrix directs the system to the relationship directory which represents the relationship held (if any) by the respective entity type pair.

The format of the POINTER record type is similar to that of the SCHEMA type. Specifically, the record is laid out as follows:

$SEQNO$, DIR1, DIR2, DIR3...DIR8

For any POINTER record with key value i, the position "DIRj" (DIRectory j) relates to the "i,j" element of the internal form pointer matrix. In this implementation said integer value is the record type number of the appropriate directory record type. As before, a record sequence number (matrix row) relates to the superordinate entity type, while the value position number (matrix column) relates to the subordinate type.

If the values of the variables ROW and COL hold the same meanings as was used above, then the following SIR statements can be used to store the appropriate record type number in the variable DIRECTOR:

```
RECORD IS POINTER (ROW)
  COMPUTE DIRECTOR = DIR1 TO DIR8 (COL)
END RECORD IS
```

Again, the "current" case instance must be 'IFORM'.

Figure C.8 shows the actual POINTER record type instances which relate to our example physical database. A circled element value denotes an implied relationship which is not shown explicitly in the conceptual schema graph. The directory record type numbers which are represented in this illustration were discussed in the previous subsection (refer back to figure C.4).

Note that only relationships of the CONTAINS and COMPRISES variety are represented in the POINTER matrix since these are the only relationship types which employ directories in their representation. Note also that the pointer matrix is processed only after the existence of the relationship is verified (as discussed above). Consequently, the system will never "see" a blank value.

$SEQNO$	DIR1	DIR2	DIR3	DIR4	DIR5	DIR6	DIR7	DIR8
1	-	-	-	-	-	-	-	-
2	-	-	-	-	-	-	-	-
3	12	-	-	-	-	-	-	-
4	(13)	(13)	(11)	-	(13)	10	(11)	11
5	-	-	-	-	-	-	-	-
6	-	-	-	-	9	-	-	-
7	(12)	(12)	-	-	(12)	-	-	-
8	(12)	(12)	-	-	12	-	-	-

◯ = IMPLIED RELATIONSHIP

"-" = BLANK

FIGURE C.8

POINTER RECORD INSTANCES

Recapitulation. In this section we have described a GERMS-based database management system (DBMS) which is implemented in SIR. SIR is an extended hierarchical (vis. true network) database system, so the design suggested in chapters 11 and 12 is modified slightly. In general, though, the design described here closely matches that proposed in the earlier chapters.

The physical database employs a number of different record types. Each record type serves as either a "data record type" or as a "pointer record type." A pointer record type is either an "index type" or a "directory type."

The complex nature of directory record keys allows for the record type to take the place of an inverted list structure. Directory rows are not of varying lengths, but record "sets" of varying sizes can be "pointed to" by a single (superordinate) data record. Thus each inverted list element is represented by a separate directory record in the SIR design.

Index record types are used to avoid physical redundancy in representing the blatant logical redundancy of "special case" entity types. The data records relating to the set of instances of a "special case" entity type are accessed by first processing the index record type.

The search is then directed to a different record type which holds the data for that and other entity type(s).

When processing a "relationship chain," the index records are used at the "start" of the chain only. If special case entities occur at "lower" chain levels, the method of processing the directory records is affected, however.

The system internal form, which is represented entirely by members of the case instance 'IFORM', involves the use of three record types: SYMBOL, SCHEMA, and POINTER. With each of these record types there is a one-to-one correspondence between record instances and logical entity types of the conceptual schema. Thus, in our example system, there are eight instances of each type. Both SCHEMA records and POINTER records are identified by their key value SEQNO. This value is the sequence number of the single logical entity type to which the record instance relates.

The SYMBOL records provide the link between entity type sequence numbers and entity type names. The SYMBOL record string key value, which is named ENAME, provides for direct access of a SYMBOL record upon specification of an entity type name. Since every SYMBOL record holds the entity sequence number upon which SCHEMA and POINTER

records are keyed, the system can subsequently access directly the respective record of either of these types. Each "column" position of the "matrix row" is likewise related to an entity sequence number.

The SYMBOL records also link the system internal form representation to a portion of the physical database. Every one of these records details the storage location and storage configuration (i.e. indexed/not indexed) of its entity type's data. If the entity's data is indexed, then SEQNO points to the index, while RECNO points to the actual data. Otherwise, both of these variables point directly to the data records.

The remainder of the logical --> physical link is achieved through the use of the POINTER records. Except for the record key, every non-blank value of a POINTER record is directed towards a relationship directory which exists within the physical database. The specific relationship represented here is indicated by the row-column position of the pointer value. There are many more pointer values than there are relationship directories since a single directory often represents a number of relationships.

With the exception of providing names to entity types, which is the duty of the SYMBOL records, the set of SCHEMA records holds the entire information content of the conceptual schema. The explicit as well as the implicit relationships of the schema are represented unambiguously. This representation is the basis of the system's "knowledge" of the conceptual schema structure. By processing these records the system can "understand" and validate the user's references to geographic relationships.

System Software

In this section we illustrate how SIR programs can be used to perform GERMS-based operations on the system whose design was described above. The intent here is to describe the basic logic of a GERMS front-end -- one which utilizes the system's knowledge of the GERMS conceptual schema. The discussion emphasizes those routines which traverse the schema network structure; concerns which are peripheral to this issue are de-emphasized.

The software described hereunder should in no way be considered as "production code." There are a number of reasons for this. First of all, the programs are incomplete as individual record attributes (the actual data values) are ignored. Also, the intricacies of how the system output is formed is not covered; nor is the diagnosing and processing of errors.

Second of all, compact software segments have been deliberately expanded to improve clarity. Thus, rather than looping through a set of statements repeatedly, the statements of the loop are repeated so that the iterations are shown explicitly. We feel that this technique better serves the illustrative purpose of this section.

Before our system can be implemented, a basal problem must be faced. It is obvious that our implementation should emphasize flexibility in its design. That is, the system user should be allowed to reference any "legitimate" set of entity types in a query. This requires that record references (i.e. PROCESS REC, RECORD IS) be allowed to be "variable" in nature.

The problem is that the use of variables for record type names or record type numbers is not allowed in these references. Thus, SIR will not permit the following:

```
COMPUTE RECTYPE = 3
PROCESS REC RECTYPE
```

The only acceptable form is

PROCESS REC 3

Note that the use of string variables for record type names is likewise not permitted.

One solution, which is clearly too clumsy to be acceptable, is to include SIR statements to allow for all possible combinations of record citations. The proper statement would be selected through the processing of a complex set of conditional branching statements.

An acceptable alternative is found through employment of a "record type substitution procedure." Such a procedure, which we call NAMESET, uses both positional and global substitution parameters (see the first section of this appendix).

The following is a source list of our procedure NAMESET:

```
00100 COMMENT Procedure NAMESET
00110 COMMENT This procedure assigns a numeric
00120         value to a global parameter name.
00130         The numeric value assigned is that of
00140         the variable whose name is passed as
00150         positional parameter <1>. The
00160         global parameter name is passed as
```

```
00170          positional parameter <2>.

00180
00190 COMMENT  ***   THIS SIR CODE IS FOR     ***
00200          *** ILLUSTRATION PURPOSES ONLY ***
00210
00220 COMMENT  ***   NOT FOR PRODUCTION USE   ***
00230
00240 IFTHEN (<1> EQ 1)
00250   GLOBAL <2> = 1
00260 ELSEIF (<1> EQ 2)
00270   GLOBAL <2> = 2
00280 ELSEIF (<1> EQ 3)
00290   GLOBAL <2> = 3
  .          .
  .          .
  .          .
00440 ELSEIF (<1> EQ 11)
00450   GLOBAL <2> = 11
00460 ELSEIF (<1> EQ 12)
00470   GLOBAL <2> = 12
00480 ELSE
00490   GLOBAL <2> = 13
```

The procedure uses two positional parameters. The value to be taken on by the first positional parameter is the name of a variable whose integer value relates to a record type number. The value of the second positional parameter is the name which is to be assigned to a global parameter whose value is that same integer represented by the first parameter. This rather complicated design deserves an example.

Consider the following SIR statements:

```
COMPUTE TYPENO = 3
CALL NAMESET ( TYPENO, RECTYPE)
```

With this CALL statement, the values assigned to the first and to the second positional parameters are "TYPENO," and "RECTYPE," respectively. TYPENO is a variable which holds integer value "3" at the time of the procedure CALL.

Given the positional parameter references in the procedure, each conditional statement is interpreted by SIR as

```
... IF (TYPENO EQ ...
```

Since TYPENO holds the value "3," statement 00280 evaluates as "true." This causes the execution of statement 00290 whose positional parameter reference is replaced by the value "RECTYPE." That is:

```
ELSEIF (TYPENO EQ 3)
  GLOBAL RECTYPE = 3
```

The ultimate result of calling the procedure is that the global parameter name "RECTYPE" is equated to the value 3 -- that very value held by the variable "TYPENO."

The purpose of this procedure is that, subsequent to the procedure CALL, each reference to the global parameter "RECTYPE" is replaced by a constant value which was previously handled as a variable; in this instance the constant value 3. Thus the statement

```
PROCESS REC <RECTYPE>
```

is subsequently interpreted as

```
PROCESS REC 3
```

The procedure NAMESET, as shown above, is designed to handle any integer value between 1 and 13 inclusive. These values relate to the record types of our physical database.

Query processing. Once a user's query statement is parsed by the system, the query must be processed. We see this processing as comprising two main tasks: query

interpretation, and data retrieval. By query interpretation we mean that portion of query processing which deals with the system internal form representation. This involves verifying the integrity of the query (not data integrity) as well as linking the logical objects referenced in the query to their physical database counterparts. The meaning of the data retrieval task is obvious.

We will discuss the software related to these two tasks separately. The query interpretation software is discussed first. The reader is reminded that the software presented below <u>is not production code.</u> It has been simplified for the purpose of illustration.

The following query processing program relates to a specific circumstance in our example system. Specifically, we deal with a query breakdown which is "nested 2-deep." Thus, if two entity types of the schema are named "X" and "Y," then we are speaking of a query of the form "GET X BY Y." Also, we assume that ALL entity instances of the type X are requested.

The query interpretation routine is entered with the assumption that the query statement is already parsed. Furthermore, it is assumed that the <u>name</u> of the superordinate entity type ("X" in the above query

template) resides in string variable "XENT." Similarly, the name of the subordinate entity type ("Y" above) is stored in string variable "YENT."

The source list is as follows:

```
00100 COMMENT This program segment interprets a query
00110         breakdown which is nested "2-deep".
00120         It is assumed that the name of the
00130         superordinate entity type is stored in
00140         string variable "XENT", and that the
00150         name of the subordinate entity type
00160         is stored in string variable "YENT."
00170         All instances of the superordinate
00180         entity type are processed.
00190
00200 COMMENT  ***   THIS SIR CODE IS FOR     ***
00210          *** ILLUSTRATION PURPOSES ONLY ***
00220
00230 COMMENT  ***   NOT FOR PRODUCTION USE   ***
00240
00250 COMMENT process internal form logical structure
00260 CASE IS 'IFORM'
00270 . SET SPECIAL1, SPECIAL2 ('FALSE')
00280 . COMMENT go to proper row of symbol table
00290           for superordinate entity type
00300 . RECORD IS SYMBOL (XENT)
00310 .   COMPUTE SUPCASE = 'ENTITY'+(FORMAT(RECNO,1))
00320 .   COMPUTE ROW = SEQNO
00330 .   CALL NAMESET (SEQNO, SUPENO)
00340 .   COMMENT check if entity type is a
00350             "special case"
00360 .   IFTHEN (RECNO NE SEQNO)
00370 .     COMPUTE SPECIAL1 = 'TRUE'
00380 .     CALL NAMESET (RECNO, SUPRNO)
00390 .   ENDIF
00400 . END RECORD IS
00410 . COMMENT go to proper row of symbol table
00420           for subordinate entity type
00430 . RECORD IS SYMBOL (YENT)
00440 .   COMPUTE SUBCASE = 'ENTITY'+(FORMAT(RECNO,1))
00450 .   COMPUTE COL = SEQNO
00460 .   CALL NAMESET (RECNO, SUBRNO)
00470 .   COMMENT check if entity type is a
```

```
00480              "special case"

00490 .    IFTHEN (RECNO NE SEQNO)
00500 .      COMPUTE SPECIAL2 = 'TRUE'
00510 .      COMPUTE SUBENAME = ENAME
00520 .    ENDIF
00530 .  END RECORD IS
00540 .  COMMENT inspect schema element for
00550            proper relationship type. if not,
00560            call error routine
00570 .  RECORD IS SCHEMA, (ROW)
00580 .    COMPUTE RELTYPE = REL1 TO REL8 (COL)
00590 .    IFNOT (RELTYPE EQ 1 OR 2) CALL RONGREL
00600 .  END RECORD IS
00610 .  COMMENT determine directory location
00620            from pointer matrix
00630 .  RECORD IS POINTER, (ROW)
00640 .    COMPUTE DIRECTOR = DIR1 TO DIR8 (COL)
00650 .    CALL NAMESET (DIRECTOR, DIRRNO)
00660 .  END RECORD IS
00670 .  COMMENT complete processing of internal
00680            form
00690 END CASE IS
```

Query interpretation deals with the system internal form, so the internal form case 'IFORM' is made "current" at statement 00260. The first task to be performed is to retrieve the required information from the symbol table. Since the SYMBOL records are "keyed" on the entity type name, statement 00300 directs the system to that single "table row" which relates to the superordinate entity type.

At statement 00310 we construct the name of that case instance of the physical database which relates to the superordinate entity type. This case name is then

stored in string variable SUPCASE (SUPerordinate CASE). The entity type sequence number is stored in variable ROW, and the record type number which relates to this sequence number is equated to global variable name "SUPENO" (SUPerordinate Entity Number) at statement 00330.

Now, if the entity type is not a "special case" entity, this record type holds the superordinate entity data. Otherwise, the record type relates to index records and the data record type must be derived. Statements 00340 through 00390 test accordingly and perform this task if appropriate. In the event that the entity is a special case type, the flag variable SPECIAL1 is set to 'TRUE,' and the global variable SUPRNO (SUPerordinate Record Number) is equated to the proper record type.

Statements 00410 through 00530 process the subordinate entity type in a similar manner. The sequence number of this entity type is stored in variable COL(umn). Note that the issue of whether or not the data is indexed is of no concern here; only the data records need be accessed.

The "special case status" of the entity does affect the processing of the directory records, however. In the event that the entity is a special case type, the flag variable SPECIAL2 is set accordingly. Also, the

subordinate entity type name (record string variable ENAME) is stored in string variable SUBENAME (SUBordinate Entity NAME).

Statements 00540 through 00600 verify the integrity of query structure by insuring that a "1" or a "2" resides in the proper row-column position of the SCHEMA "matrix." This position is determined by the values of the variables ROW and COL which relate to the sequence numbers of the superordinate and the subordinate entity types, respectively.

If a proper schema value is not found (i.e. statement 00590 evaluates as "true"), then a CONTAINS or a COMPRISES relationship does not hold, in the proper direction, between the entity type pair in question. This situation triggers the execution of the procedure RONGREL (wRONG RELationship). This procedure is not described here, but its purpose is obvious -- it causes the query to be rejected.

Next, if the query is found to be valid, the location of the appropriate relationship directory is located. Statements 00610 through 00660 accomplish this. The type number of the directory record type is stored in the appropriate row-column position of the POINTER "matrix." Once located, this type number is equated to

global parameter name DIRRNO (DIRectory Record Number) at statement 00650.

Finally, statement 00690 indicates that the system internal form (case 'IFORM') is no longer needed for the processing of this query. We have extracted all of the information that is required to perform the query interpretation phase of query processing.

Before we begin our discussion of the data retrieval phase of query processing, we will add that the extensions that are required to allow the above program to handle "n-deep nested" queries are minimal. Essentially, those statements which have to do with the subordinate type and with the relationship (i.e. statements 00410 through 00660) are placed in a loop which is processed once for each level of nesting (obviously, some means of differentiating the variable names is also required). This feature is omitted here for the sake of simplicity.

The task of data retrieval demands that the record processing blocks be nested within one another. In the two-entity query discussed here (X BY Y), the subordinate data record processing block (record type SUBRNO) is nested within the directory record processing block (type DIRRNO). This, in turn, is nested within the superordinate index and/or data record processing block

(types SUPENO, SUPRNO). The processing block structure of "deeper nested" query formats is obvious.

The source list of the data retrieval program segment is as follows:

```
00700 COMMENT This program segment performs
00710         the data retrieval portion of query
00720         processing. Terms enclosed in "<  >"
00730         are substituted with record type
00740         numbers before processing.
00750
00760 COMMENT  ***   THIS SIR CODE IS FOR     ***
00770          *** ILLUSTRATION PURPOSES ONLY ***
00780
00790 COMMENT  ***   NOT FOR PRODUCTION USE   ***
00800
00810 CALL *
00820 COMMENT begin with superordinate entity
00830 CASE IS SUPCASE
00840 . PROCESS REC <SUPENO>
00850 .   COMPUTE KEY1 = KEY
00860 .   COMMENT check if entity type is a
00870                 "special case"
00880 .   IFTHEN (SPECIAL1 EQ 'TRUE')
00890 .     COMMENT if yes, move to and
00900                   process data record
00910 .     RECORD IS <SUPRNO>, (KEY1)
00920 .       CALL WRITPROC
00930 .     END RECORD IS
00940 .   ELSE
00950 .     COMMENT otherwise, process data record
00960 .     CALL WRITPROC
00970 .   ENDIF
00980 .   COMMENT check if subordinate entity is a
00990                 "special case"
01000 .   IFTHEN (SPECIAL2 EQ 'TRUE')
01010 .     COMMENT if yes, process reduced set of
01020                   directory siblings
01030 .     PROCESS REC <DIRRNO>, WITH (KEY1, SUBENAME)
01040 .       COMPUTE KEY2 = PTR
01050 .       COMMENT follow directory pointer
01060 .       CASE IS SUBCASE
01070 .         RECORD IS <SUBRNO>, (KEY2)
```

```
01080 .           CALL WRITPROC

01090 .         END RECORD IS
01100 .       END CASE IS
01110 .     END PROCESS REC
01120 .   ELSE
01130 .     COMMENT otherwise, process entire set
01140               of directory siblings
01150 .     PROCESS REC <DIRRNO>, WITH (KEY1)
01160 .       COMPUTE KEY2 = PTR
01170 .       COMMENT follow directory pointer
01180 .       CASE IS SUBCASE
01190 .         RECORD IS <SUBRNO>, (KEY2)
01200 .           CALL WRITPROC
01210 .         END RECORD IS
01220 .       END CASE IS
01230 .     END PROCESS REC
01240 .   ENDIF
01250 . END PROCESS REC
01260 . COMMENT complete external superordinate
01270           entity block
01280 END CASE IS
```

The first operation, statement 00830, directs the system to the physical database (vis. system internal form). This statement makes current that case instance which relates to the superordinate entity type. Next, the "outside" record processing block, that which relates to this same superordinate entity, is entered.

The conditional block which comprises statements 00860 through 00970 dictates that the appropriate action be taken depending on whether or not the entity type in question is a "special case." If the entity type is not a special case, then the current record is a data record and can be processed directly.

If, on the other hand, it <u>is</u> a special case, then the current record is an <u>index</u> record; its data record counterpart must be made current (statement 00910) before processing is carried out.

The processing of a data record involves the formatting and writing of the requested record attributes to the output file. This task is peripheral to our concerns here, so we will assume that the procedure WRITPROC (WRITe PROCedure) performs this job properly. This procedure is not discussed further.

The processing of subordinate data records is affected by the processing of directory records, which in turn is affected by the "special case status" of the subordinate entity type. Consequently, these two nested processing blocks are themselves nested within a two-branch conditional block (statements 00980 through 01240).

If the subordinate entity type <u>is</u> a special case entity, then a reduced set of directory siblings is considered. This reduced set is specified by stating <u>two</u> sort variables; besides the key of the superordinate entity instance, the subordinate entity type <u>name</u> is matched (see statement 01030).

For each directory record considered, the pointer value (variable PTR) is "followed." First, however, that case instance which relates to the subordinate entity type is made current (statement 01060). Following this, the subordinate data record which is pointed to by the directory record is processed. Again, the procedure WRITPROC accomplishes this.

If the subordinate entity type is not a special case type, the basic steps remain the same -- the only difference being that a larger set of directory entries is processed (compare statement 01150 with statement 01030). These steps relate to program statements 01130 through 01230.

Statement 01250 signals the logical terminus of the "external" (superordinate) record processing block. Following the exit from this block the superordinate entity case instance need no longer be considered. This is represented by statement 01280 which completes the query processing program.

A semantic integrity checker. In chapter 13 we described the logic of a subsystem which is used to verify the integrity of the objects of the physical database. Sets of data are aggregated and the resulting values

compared to those of related data. Based on the meaning of the data there can be derived various relations (e.g. EQ, GT) which should hold between the compared values. We call these semantic relations since they are derived from the data meaning. The subsystem is called a semantic integrity checker for a similar reason.

In the following paragraphs we show how the semantic integrity checker can be programmed to serve as a useful accessory to our SIR implementation of the GERMS-based system. The logic of the programs presented here varies slightly from that presented earlier. The differences are due to the idiosyncrasies of SIR. For the most part, however, the programs adhere to the earlier described structure.

The semantic integrity checker comprises a main routine and a number of CALLable procedures, some of which are not described in detail. When the main routine is executed, the integrity of the entire physical database is checked. Admittedly, this checking constitutes an expensive undertaking. The relatively static nature of geopolitical statistical data insures that the task need not be performed often, however.

The main routine, whose structure is hardly complex, is as follows:

```
00100 RETRIEVAL
00110 COMMENT This program is the main routine
00120         of a semantic integrity checker.
00130         Each schema element is inspected
00140         for the presence of a CONTAINS or
00150         a COMPRISES relationship. Upon
00160         finding such, the integrity of the
00170         respective data objects is verified.
00180
00190 COMMENT  ***   THIS SIR CODE IS FOR     ***
00200          *** ILLUSTRATION PURPOSES ONLY ***
00210
00220 COMMENT  ***   NOT FOR PRODUCTION USE   ***
00230
00240 COMMENT process internal form logical structure
00250 CASE IS 'IFORM'
00260 . COMMENT process each "row" of schema "matrix"
00270 . PROCESS REC SCHEMA
00280 .   COMPUTE ROW  = SEQNO
00290 .   COMMENT loop over matrix "columns"
00300 .   FOR COL = 1, 8
00310 .     COMPUTE RELTYPE = REL1 TO REL8 (COL)
00320 .     COMMENT inspect relationship type and
00330                 trigger appropriate action
00340 .     IF (RELTYPE EQ 1 OR 2) CALL SUMCHEK
00350 .   END FOR
00360 . END PROCESS REC
00370 . COMMENT complete processing of internal form
00380 END CASE IS
00390 END RETRIEVAL
```

This program directs the system through a "row-wise" scan of the internal form schema representation. The "current" case instance is therefore 'IFORM,' as specified in statement 00250. The major portion of the program is

nested within a SCHEMA record type processing block (statements 00260 through 00360). Every SCHEMA record is considered in a separate iteration through this block.

For each SCHEMA record (matrix row), every row element is considered in a separate iteration of a "column loop" (statements 00300 through 00350). The integer value of the schema element, which describes the nature of the relationship (if any) held between the respective entity type pair, is stored in variable RELTYPE (RELationship TYPE).

Now, if the schema element value is a "1" or a "2," then the respective entity type pair is linked by a CONTAINS, or by a COMPRISES relationship, respectively. If such is the case, the data integrity can be verified.

The procedure SUMCHEK, which performs this verification, is CALLed in the event that one of these relationship types is found (statement 00340). Otherwise, the "next" schema element is considered. When each column of each row of the matrix is inspected, and the proper action taken, the program terminates.

The logic of procedure SUMCHEK is quite similar to that of the query processing software described earlier (data values are compared rather then printed). For this reason our discussion will concentrate, for the most part, on those features which are unique to SUMCHEK.

The following is a source list of the procedure.

```
00100 COMMENT Procedure SUMCHEK
00110 COMMENT This procedure is CALLed in
00120         the event that a CONTAINS or
00130         a COMPRISES relationship is
00140         detected in the schema matrix.
00150         The purpose is to verify that
00160         the proper semantic relations
00170         hold among the related data
00180         objects.
00190
00200 COMMENT  ***   THIS SIR CODE IS FOR     ***
00210          *** ILLUSTRATION PURPOSES ONLY ***
00220
00230 COMMENT  ***   NOT FOR PRODUCTION USE   ***
00240
00250 . COMMENT continue processing internal form
00260           case 'IFORM'
00270 . SET SPECIAL1, SPECIAL2 ('FALSE')
00280 . SET FOUND (0)
00290 . COMMENT search symbol "table" until both
00300           row and column records are found
00310 . PROCESS REC SYMBOL
00320 .   IF (FOUND EQ 2) EXIT RECORD
00330 .   COMMENT check if record relates to
00340             schema row
00350 .   IFTHEN (SEQNO EQ ROW)
00360 .     COMMENT if yes, prepare for processing
00370               of superordinate records
00380 .     COMPUTE SUPCASE = 'ENTITY'+(FORMAT(RECNO,1))
00390 .     CALL NAMESET (SEQNO, SUPENO)
00400 .     COMMENT check if entity type is a
00410               "special case"
00420 .     IFTHEN (RECNO NE SEQNO)
00430 .       COMPUTE SPECIAL1 = 'TRUE'
00440 .       CALL NAMESET (RECNO, SUPRNO)
00450 .     ENDIF
00460 .     COMMENT record that a match has been
00470               found and discard this record
00480 .     COMPUTE FOUND = FOUND+1
00490 .     NEXT RECORD
00500 .   ENDIF
00510 .   COMMENT check if record relates to schema
00520             column
00530 .   IFTHEN (SEQNO EQ COL)
```

```
00540 .     COMMENT if yes, prepare for processing

00550               of subordinate entity records
00560 .     COMPUTE SUBCASE = 'ENTITY'+(FORMAT(RECNO,1))
00570 .     CALL NAMESET (RECNO, SUBRNO)
00580 .     COMMENT check if entity type is a
00590               "special case"
00600 .     IFTHEN (RECNO NE SEQNO)
00610 .       COMPUTE SPECIAL2 = 'TRUE'
00620 .       COMPUTE SUBENAME = ENAME
00630 .     ENDIF
00640 .     COMMENT record that a match was found
00650 .     COMPUTE FOUND = FOUND+1
00660 .   ENDIF
00670 . END PROCESS REC
00680 . COMMENT determine directory location from
00690             pointer matrix
00700 . RECORD IS POINTER, (ROW)
00710 .   COMPUTE DIRECTOR  = DIR1 TO DIR8 (COL)
00720 .   CALL NAMESET (DIRECTOR, DIRRNO)
00730 . END RECORD IS
00740 . COMMENT move to  physical database and
00750             begin with superordinate entity
00760             type
00770 . CASE IS SUPCASE
00780 .   PROCESS REC <SUPENO>
00790 .     COMPUTE KEY1 = KEY
00800 .     COMMENT check if entity type is a
00810               "special case"
00820 .     IFTHEN (SPECIAL1 EQ 'TRUE')
00830 .       COMMENT if yes, move to and process
00840                 data records
00850 .       RECORD IS <SUPRNO>, (KEY1)
00860 .         MOVE VARS ALL
00870 .       END RECORD IS
00880 .     ELSE
00890 .       COMMENT otherwise, process data record
00900 .       MOVE VARS ALL
00910 .     ENDIF
00920 .     COMMENT check if subordinate entity is a
00930               "special case"
00940 .     IFTHEN (SPECIAL2 EQ 'TRUE')
00950 .       COMMENT if yes, process reduced set of
00960                 directory siblings
00970 .       PROCESS REC <DIRRNO>, WITH (KEY1, SUBENAME)
00980 .         COMPUTE KEY2 = PTR
00990 .         COMMENT follow directory pointer
01000 .         CASE IS SUBCASE
01010 .           RECORD IS <SUBRNO>, (KEY2)
```

```
01020 .              CALL AGGREGAT

01030 .            END RECORD IS
01040 .          END CASE IS
01050 .        END PROCESS REC
01060 .      ELSE
01070 .        COMMENT otherwise, process entire set of
01080                  directory records
01090 .        PROCESS REC <DIRRNO>, WITH (KEY1)
01100 .          COMPUTE KEY2 = PTR
01110 .          COMMENT follow directory pointer
01120 .          CASE IS SUBCASE
01130 .            RECORD IS <SUBRNO>, (KEY2)
01140 .              CALL AGGREGAT
01150 .            END RECORD IS
01160 .          END CASE IS
01170 .        END PROCESS REC
01180 .      ENDIF
01190 .      COMMENT verify semantic relation
01200                depending on type of relationship
01210 .      IFTHEN (RELTYPE EQ 1)
01220 .        CALL CONTCHK
01230 .      ELSE
01240 .        CALL COMPCHK
01250 .        ENDIF
01260 .    END PROCESS REC
01270 .    COMMENT exit physical database
01280 . END CASE IS
```

Note that the entire procedure is nested within the 'IFORM' case processing block of the CALLing program (main routine). This is despite the fact that case instances of the physical database are considered.

As the procedure begins we continue to process the internal form representation. This is required since additional information about the physical database must be derived from the SYMBOL "table," and from the POINTER "matrix."

One minor complexity is that the program holds entity type sequence numbers (variables ROW, COL) which relate to the row-column pair of a schema element, but the SYMBOL records are keyed on entity type names. Consequently, the SYMBOL records must be inspected one at a time until the proper two records are found. This task relates to the SYMBOL record type processing block which comprises statements 00290 through 00670.

The variable FOUND is used to hold the count of SYMBOL record "hits" which have been encountered. When this count reaches 2 (both row and column data found), as detected at statement 00320, the SYMBOL table processing is terminated.

The conditional block which comprises statements 00330 through 00500 deals with the situation where the current SYMBOL record relates to the schema row in question. When such a "match" is found, the record is that of the superordinate entity type in the CONTAINS or COMPRISES relationship being verified.

The information in the SYMBOL record is used to prepare for processing of the physical database as was described before. That is, (1) the superordinate case instance name is constructed, and (2) the superordinate index and/or data record type numbers are prepared. Note

that statement 00490 signals that the "next" SYMBOL record can be considered as soon as the "row relevant" information has been derived. This is because a single SYMBOL record will not "match" both the schema row and the schema column -- an entity type is never related to itself.

The conditional block made up of statements 00510 through 00610 serves a similar purpose for the instance where the current SYMBOL record relates to the schema column in question. In this block the processing of the subordinate data (of the physical database) is prepared for. Thus, (1) the name of the subordinate case instance is constructed, and (2) the subordinate data record type number is prepared. Also, if the type is a special case, its name is stored for later use.

Following completion of SYMBOL "table" processing, the only information which remains to be taken from the internal form representation has to do with locating the proper relationship directory records. Recall that the location is stored as the integer value of an element of the POINTER "matrix." Statements 00680 through 00730 recover this value and prepare the directory record type number accordingly.

At this point, processing is directed towards the physical database; the system internal form is (temporarily) discarded. The record processing blocks which deal with the physical database hold the same nested structure as do the blocks of the aforementioned query processing data retrieval program. Thus, the subordinate entity record processing block is nested within the directory processing block which, in turn, is nested within the superordinate entity record processing block. The reader will notice that the "intra-block" logic is also very much the same.

The external record processing block involves all record instances relating to the superordinate entity type (record type SUPENO in our software). For each of these instances the variable KEY1 is equated to the record key value at statement 00790. If this record type is used as an index, as determined at statement 00820, then the respective data record is made current before actual data items are considered. Otherwise, the initial record is processed as is (it is a data record). Statements 00830 through 00870 deal with the former situation. Statements 00980 and 00900 deal with the latter.

Processing here involves the use of the MOVE VARS statement. This statement instructs that all data values be "moved" to a buffer area so that they may be referenced later. It is these values which are to be compared with the aggregation of their subordinate counterparts.

For each superordinate data record processed, the set of directory records (record type DIRRNO) which are pointed to in the relationship must be considered. The size of this set depends on the "special case status" of the subordinate entity type in question. This status is determined by conditional statement 00940.

If the entity type is a special case, the directory records are a subset of those considered in the alternative situation. The record sets which relate to these two situations are defined by statements 00970 and 01090, respectively. Note that in the former case a second sort variable, the subordinate entity type name, is specified.

Regardless of the special case status, the method of processing a directory record and the subordinate data record which it points to is the same. First, the directory pointer is followed. This demands that a new case instance be made current (e.g. statement 01120). The single data record which is pointed to is an instance of the subordinate record type SUBRNO.

Next, after said record instance is accessed, the procedure AGGREGAT is called. This procedure is not detailed herein, but its function should be apparent from its name. Essentially, the procedure forms the aggregate of each individual record data item. The aggregation is performed "across" the subordinate data records of a relationship instance, but "within" the processing block of the superordinate entity. Therefore, following the completion of the "internal" (directory and subordinate) record processing blocks, the intra-relationship aggregate values are available to the system. Thus, the only task that remains is to verify that the proper semantic relations exist.

Substantiation of the semantic relations is performed by SUMCHEK statements 01190 through 01250. Recall that the invocation of this procedure is triggered by the discovery that a schema relationship type, as represented by the integer value of variable RELTYPE, relates to a CONTAINS or to a COMPRISES relationship (integer value "1" or "2," respectively). Now, depending on the relationship type being checked, a certain semantic relation must be verified. Specifically, in the case of a COMPRISES relationship, the aggregation of the subordinate data values should exactly equal its superordinate

counterpart. In the CONTAINS situation on the other hand, the aggregate value should be compared via a less than or equal to relation.

The relationship type is determined by conditional statement 01210. If the relationship is of the CONTAINS variety (RELTYPE = 1), then procedure CONTCHK (CONTains CHecK) is called. Otherwise, procedure COMPCHK (COMPrises CHecK) is called. These "checking" procedures are not described in detail because their purpose and structure are obvious. It is in these procedures that the superordinate data values which were previously "moved" are utilized. In the event that the proper semantic relation is found to be lacking, an appropriate error message is displayed.

We will reiterate that the checking of semantic relations occurs within the superordinate record processing block. The checking therefore relates to a single relationship instance.

The nadir of the superordinate processing block occurs at statement 01260. At this point the physical database is no longer of concern and is therefore exited (statement 01280). This marks the end of the procedure, so the initial CALLing routine is returned to.

The internal form representation case instance is again made current, and the inspection of the schema elements continues as before. After each schema element is considered, and the appropriate checks made, the main semantic integrity routine terminates.

FOOTNOTES

[1] Alternatively, the entire matrix can be represented within a single record which holds sixty-four (N*N) values, not including the record key. If the matrix is represented in row-major order, then matrix element "i,j" will reside at value position (N * (i - 1) + j). Thus, given any matrix row-column index pair, the appropriate element value can be located. If the first record value holds the record key, then the base address should be adjusted by adding "1" to this formula.

BIBLIOGRAPHY

[ABRI74] Abrial, J. R. "Data semantics," in Data Base Management (Klimbie, J. W., and Koffeman, K. L., eds.), pp. 1-59. North-Holland, Amsterdam, 1974.

[ADIB76] Adiba, M., Delobel, C., and Leonard, M. "A unified approach for modeling data in logical data base design," in Modeling in Data Base Management Systems (Nijssen, G. M., ed.), pp. 311-338. North-Holland, Amsterdam, 1976.

[ASHB64] Ashby, W. R. An Introduction to Cybernetics. Methuen & Co. Ltd., London, 1964.

[BACH79] Bachman, C. W., and Daya, M. "The role concept in data models," Proc. 3rd Int. Conf. Very Large Data Bases, pp. 464-476, 1977.

[BALD79] Baldissera, C., Ceri, S., Pelagatti, G., and Bracchi, G. "Interactive specification and formal verification of user's views in data base design," Proc. 5th Int. Conf. Very Large Data Bases, pp. 262-272, 1979.

[BALZ79] Balzer, R. "An implementation methodology for semantic data base models," Proc. E-R Conference pp. 415-425, 1979.

[BEER78] Beeri, C., Bernstein, P. A., and Goodman, N. "A sophisticate's introduction to database normalization theory," Proc. 4th Int. Conf. Very Large Data Bases, pp. 113-124, 1978.

[BELL81] Bell, J. R. "Future directions in computing," Computer Design. 20, pp. 95-102, 1981.

[BENC76] Benci, E., Bodart, F., Bogaert, H., and Cabanes, A. "Concepts for the design of a conceptual schema," in Modeling in Data Base Management Systems (Nijssen, G. M., ed.), pp. 181-200. North-Holland, Amsterdam, 1976.

[BERN81] Bernstein, P. A., and Goodman, N. "Concurrency control in distributed database systems," ACM Comp. Surveys. 13, pp. 185-222, 1981.

[BILL78] Biller, H., and Neuhold, E. J. "Semantics of data bases: The semantics of data models," Inf. Syst. 3, pp. 11-30, 1978.

[BONC81] Bonczek, R. H., et al. "A generalized decision support system using predicate calculus and network data base management," Operations Research 29, pp. 263-281, 1981.

[BORK78] Borkin, S. A. "Data model equivalence," Proc. 4th Int. Conf. Very Large Data Bases, pp. 526-534, 1978.

[BRAC79] Brachman, R. J. "On the epistemological status of semantic networks," in Associative Networks (Findler, N., ed.), pp. 3-50. Academic Press, New York, 1979.

[BRAY82] Bray, O. H. Distributed Database Management Systems. Lexington Books, Lexington, MA, 1982.

[BROO82] Brooks, C. H. P., et al. Information Systems Design. Prentice-Hall of Australia Pty Ltd., Netley, South Australia, 1982.

[BURC83] Burch, J. G., Strater, F. R., and Grudnitski, G. Information Systems: Theory and Practice, 3rd ed. John Wiley and Sons Inc., Santa Barbara, CA, 1983.

[CARL76] Carlson, C. R., and Kaplan, R. S. "A generalized access path model and its application to a relational data base system," Proc. ACM SIGMOD, pp. 143-145, 1976.

[CENS80] Census of Population and Housing, 1980 -- Summary Tape File 1: 1978 Richmond Dress Rehersal Technical Documentation / prepared by the Data User Services Division, Bureau of the Census. -- Washington: The Bureau, 1980.

[CHAM81] Chamberlin, D. et al. "A history and evaluation of system R," Commun. ACM. 24, pp. 632-647, 1981.

[CHEN76] Chen, P. P. "The entity-relationship model: Toward a unified view of data," ACM Trans. Database Syst. 1, pp. 9-36, 1976.

[CHEN77] Chen, P. P. "The entity-relationship model: A basis for the enterprise view of data," Proc. AFIPS NCC 46, pp. 77-84, 1977.

[CHEN80] Chen, P. P. (ed.) Entity-Relationship Approach to Systems Analysis and Design. North-Holland, Amsterdam, 1980.

[CHIN81] Chin, F., and Ozsoyoglu, G. "Statistical database design," ACM Trans. Database Syst. 6, pp. 113-139, 1981.

[CHUR56] Church, A. An Introduction to Mathematical Logic I. Princeton Univ. Press, Princeton, NJ, 1956.

[CLEM81] Clemons, E. H. "Design of an external schema facility to define and process recursive structures," ACM Trans. Database Syst. 6, pp. 295-311, 1981.

[CODA62] CODASYL Development Committee "An information algebra," in System Analysis Techniques (Couger, J. D., and Knapp, R. W., eds.), pp. 234-258. John Wiley and Sons, New York, NY, 1962.

[CODD70] Codd, E. F. "A relational model of data for large shared data banks," Commun. ACM 13, pp. 377-387, 1970.

[CODD72] Codd, E. F. "Further normalization of the data base relational model," in Data Base Systems, Courant Comput. Sci. Symp. 6th (Rustin, R., ed.), pp. 33-64. Prentice-Hall, Englewood Cliffs, NJ, 1972.

[CODD74] Codd, E. F. "Seven steps to RENDEZVOUS with the casual user," in Data Base Management (Klimbie, J. W., and Koffeman, K. L., eds.), pp. 179-199. North-Holland, Amsterdam, 1974.

[CODD79] Codd, E. F. "Extending the database relational model to capture more meaning," ACM Trans. Database Syst. 4, pp. 397-434, 1979.

[CREN80] Crenner, J. J. "Productivity and the Information Explosion," The Information Manager, su., pp. 15, 16, 33; 1980.

[DALE77] Dale, A. G., and Dale, N. B. "Main schema-external schema interaction in hierarchically organized data bases," Proc. ACM SIGMOD, pp. 102-110, 1977.

[DATE81] Date, C. J. An Introduction to Database Systems, 3rd ed. Addison-Wesley, Reading, MA, 1981.

[DELI79] Deliyanni, A., and Kowalski, R. A. "Logic and semantic networks," Commun. ACM 22, pp. 184-192, 1979.

[DENN81] Denning, D. E., and Sacco, G. M. "Timestamps in key distribution protocols," Commun. ACM. 24, pp. 533-536, 1981.

[DIJK72] Dijkstra, E. W. "Notes on structured programming," in Structured Programming (Dahl, O. -J., Dijkstra, E. W., and Hoare, C. A. R., eds.), pp. 1-82. Academic Press, New York, 1972.

[DIJK76] Dijkstra, E. W. A Discipline of Programming. Prentice-Hall, Englewood Cliffs, NJ, 1976.

[ELSO73] Elson, M. Concepts of Programming Languages. Science Research Associates, Chicago, IL, 1973.

[ESWA75] Eswaran, K. P., and Chamberlin, D. D. "Functional specifications of a subsystem for data base integrity," Proc. 1st Int. Conf. Very Large Data Bases, pp. 48-68, 1975.

[FRY76] Fry, J. P., and Sibley, E. H. "The evolution of database management systems," ACM Comput. Surv. 8, pp. 7-42, 1976.

[GRAY81] Gray, J. et al. "The recovery manager of the system R database manager," ACM Comp. Surveys. 13, pp. 223-242, 1981.

[HALL76] Hall, P., Owlett, J., and Todd, S. J. P. "Relations and entities," in Modeling in Data Base Management Systems (Nijssen, G. M., ed.), pp. 201-220. North-Holland, Amsterdam, 1976.

[HAMM75] Hammer, M. M., and McLeod, D. J. "Semantic integrity in a relatinal data base system," Proc. 1st Int. Conf. Very Large Data Bases, pp. 25-47, 1975.

[HAMM76] Hammer, M. M. "Data abstractions for data bases," Proc. Conf. on Data: Abstraction, Definition and Structure, ACM FDT8(2), pp. 58-9, 1976.

[HAMM81] Hammer, M., and McLeod, D. "Database description with SDM: A semantic database model," ACM Trans. Database Syst. 6, pp. 351-386, 1981.

[HANN71] Hanna, S. C., and Saber, J. C. Sets and Logic. Richard D. Irwin, Inc., Homewood, IL, 1971.

[HASE77] Haseman, W. D., and Whinston, A. B. Introduction to Data Management. Richard D. Irwin, Homewood, IL, 1977.

[HEND77] Hendrix, G. G. "Some general comments on semantic networks," Proc. 5th Int. Joint Conf. on Artificial Intelligence, pp. 984-985, 1977.

[HEND78] Hendrix. G. G., et al. "Developing a natural language interface to complex data," ACM Trans. Database Syst. 3, pp. 105-147, 1978.

[HOAR72] Hoare, C. A. R. "Notes on data structuring," in Structured Programming (Dahl, O.-J., Dijkstra, E. W., and Hoare, C. A. R., eds.), pp. 83-174. Academic Press, New York, NY, 1972.

[HOAR81] Hoare, C. A. "The emperor's old clothes," Commun. ACM. 24, pp. 75-83, 1981.

[HONG81] Hong, Y. C., and Su, S. Y. W. "Associative hardware and software techniques for integrity control," ACM Trans. Database Syst. 6, pp. 416-440, 1981.

[HORO76] Horowitz, E., and Sahni, S. Fundamentals of Data Structures. Computer Science Press, Potomac, MD, 1976.

[HUBB81] Hubbard, G. U. Computer Assisted Data Base Design. Van Nostrand Reinhold Co., London, 1981.

[HUSS81] Hussain, D., and Hussain, K. M. Information Processing Systems for Management. Richard D. Irwin Inc., Homewood, IL, 1981.

[INMO81] Inmon, W. H. Effective Data Base Design. Prentice-Hall Inc., Englewood Cliffs, NJ, 1980.

[IOSS80] Iossiphidis, J. "A translator to convert the DDL of ERM to the DDL of System 2000," in Entity-Relationship Approach to Systems Analysis and Design (Chen, P. P., ed.), pp. 477-504. North-Holland Amsterdam, 1980.

[JAME59] James, G., and James, R. C. (eds.). Mathematics Dictionary. Van Nostrand, Princeton, NJ, 1959.

[KAHN76] Kahn, B. K. "A method for describing information required by the database design process," Proc. ACM SIGMOD, pp. 53-64, 1976.

[KAPL80] Kaplan, C. P., and Valey, T. L. Census'80: Continuing the Factfinder Tradition. U.S. Department of Commerce Bureau of the Census, Washington, DC, 1980.

[KEME59] Kemeny, J. G., et al. Finite Mathematical Structures. Prentice-Hall Inc., Englewood Cliffs, NJ, 1959.

[KENT79] Kent, W. "Limitations of record-oriented information models," ACM Trans. Database Syst. 4, pp. 107-131, 1979.

[KERL64] Kerlinger, F. N. Foundations of Behavioral Research. Holt Reinhart, Winston, New York, 1964.

[KERS76] Kerschberg, L., Klug, A., and Tsichritzis, D. C. "A taxonomy of data models," in Systems for Large Data Bases (Lockemann, P. C., and Neuhold, E. J., eds.), pp.43-64. North-Holland, Amsterdam, 1976.

[KLUG77] Klug, A., and Tsichritzis, D. C. "Multiple view support within the ANSI/SPARC framework," Proc. 3rd Int. Conf. Very Large Data Bases, pp. 477-488, 1977.

[KNUT68] Knuth, D. E. The Art of Computer Programming 1, Fundamental Algorithms. Addison-Wesley, Reading, MA, 1968.

[KOHL81] Kohler, W. H. "A survey of techniques for synchronization and recovery in decentralized computer systems," ACM Comp. Surveys. 13, pp. 149-184, 1981.

[KROE77] Kroenke, D. Database Processing: Fundamentals, Modeling, Applications. Science Research Associates, Palo Alto, CA, 1977.

[LANG80] Langefors, B. "Infological models and information user views," Inf. Syst. 5, pp. 17-32, 1980.

[LEE76] Lee, J. A. N., and Ralston, A. "Syntax, semantics, and pragmatics," in Encyclopedia of Computer Science (Ralston, A., ed.), pp. 1389-1390. Van Nostrand Reinhold Co., New York, 1976.

[LEHM81] Lehman, P. L., and Bing, S. Y. "Efficient locking for concurrent operations on B-trees," ACM Trans. Database Syst. 6, pp. 650-670, 1981.

[LIEN81] Lien, Y. E. "Hierarchical schemata for relational databases," ACM Trans. Database Syst. 6, pp. 48-69, 1981.

[LISK74] Liskov, B. H., and Zilles, S. N. "Programming with abstract data types," Proc. Symp. on Very High Level Languages, SIGPLAN Notices 9(4), pp. 50-59, 1974.

[LISK75] Liskov, B. H., and Zilles, S. N. "Specification techniques for data abstractions," Proc. Int. Conf. on Reliable Software, ACM SIGPLAN Notices 10(6), pp. 72-87, 1975.

[LOCK79] Lockemann, P. C., Mayr, H. C., Weil, W. H., and Wohlleber, W. H. "Data abstractions for database systems," ACM Trans. Database Syst. 4, pp. 60-75, 1979.

[LUM79] Lum, V. Y., et al. "1978 New Orleans data base design workshop report," Proc. 5th Int. Conf. Very Large Data Bases, pp. 328-350, 1979.

[LUND81] Lundeberg, M., Goldkuhl, G., and Nilsson, A. Information Systems Development -- A Systematic Approach. Prentice-Hall, Englewood Cliffs, NJ, 1981.

[MARC81] March, S. T., et al. "Frame memory: A storage architecture to support rapid design and implementation of efficient databases," ACM Trans. Database Syst. 6, pp. 441-463, 1981.

[MART73] Martin, J. Security, Accuracy, and Privacy in Computer Systems. Prentice-Hall, Inc., Englewood Cliffs, NJ, 1973.

[MART75] Martin, J. Computer Data-Base Organization. Prentice-Hall, Englewood Cliffs, NJ, 1975.

[MART76] Martin, J. Principles of Data-Base Management. Prentice-Hall Inc., Englewood Cliffs, NJ, 1976.

[MEND79] Mendelzon, A. O., and Maier, D. "Generalized mutual dependencies and the decomposition of database relations," Proc. 5th Int. Conf. Very Large Data Bases, pp. 75-82, 1979.

[MERT74] Merten, A. G., and Fry, J. P. "A data description language approach to file translation," Proc. ACM SIGMOD, pp. 191-205, 1974.

[MINK77] Minker, J. "An experimental relational data base system based on logic," in Logic and Data Bases, (Gallaire, H., and Minker, J., eds.), pp. 107-147, 1977.

[MINS74] Minsky, N. "On interaction with data bases," Proc. ACM SIGFIDET Workshop on Data Description, Access, and Control, 1974.

[MOYE73] Moyer, D. D., and Fisher, K. P. Land Parcel Identifiers for Information Systems. American Bar Foundation, Chicago, IL, 1973.

[MYLO75] Mylopoulos, J., et al. "TORUS -- a natural language understanding system for data management," IJCAI, 4, 1974.

[NAVA78] Navathe, S. B., and Schkolnick, M. "View representation in logical data base design," Proc. ACM SIGMOD, pp. 144-156, 1978.

[NAVA80a] Navathe, S. B. "Schema analysis for database restructuring," ACM Trans. Database Syst. 5, pp. 157-184, 1980.

[NAVA80b] Navathe, S. B. "An intuitive approach to nornmalize network structured data," Proc. 6th Int. Conf. Very Large Data Bases, pp. 350-358, 1980.

[NIJS77] Nijssen, G. M. "Current issues in conceptual schema," in Architecture and Models in Data Base Management Systems (Nijssen, G. M., ed.), pp. 31-65. North-Holland, Amsterdam, 1977.

[PAGE78] Page, E. S., and Wilson, L. B. Information Representation and Manipulation in a Computer. Cambridge University Press, London, 1978.

[POHL81] Pohl, I., and Shaw, A. The Nature of Computation: An Introduction to Computer Science. Computer Science Press, Rockville, MD, 1981.

[REIS81] Reisner, P. "Human factors studies of database query languages: A survey and assessment," ACM Comput. Surv. 13, pp. 13-31, 1981.

[REIT78] Reiter, R. "On closed world databases," in Logic and Data Bases (Gallaire, H., and Minker, J., eds.), pp. 55-76. Plenum Press, New York, 1978.

[RICA81] Ricart, G., and Agrawala, A. "An optimal algorithm for mutual exclusion in computer networks," Commun. ACM. 24, pp. 9-17, 1981.

[ROBI80] Robinson, B. N., et al. SIR Scientific Information Retrieval User's Manual Version 2. SIR Inc., Evanston, IL, 1980.

[ROSS78] Ross, R. G. Data Base Systems: Design, Implementation, and Management. Amacom, New York, 1978.

[ROUS75] Roussopoulos, N., and Mylopoulos, J. "Using semantic networks for data base management," Proc. 1st Int. Conf. Very Large Data Bases, pp. 144-172, 1975.

[SAKA80] Sakai, H. "A unified approach to the logical design of a hierarchical data model," in Entity-Relationship Approach to Systems Analysis and Design (Chen, P. P., ed.), pp. 61-74. North-Holland, Amsterdam, 1980.

[SCHM75] Schmid, H. A., and Swenson, J. R. "On the semantics of the relational data model," Proc. ACM SIGMOD, pp. 211-223, 1975.

[SCHN76] Schneider, L. S. "A relational view of the data independent accessing model," Proc. ACM SIGMOD, pp. 75-90, 1976.

[SCHU76] Schubert, L. K. "Extending the expressive power of semantic networks," Artificial Intelligence 7, pp. 163-198, 1976.

[SENK75] Senko, M. E. "The DDL in the context of a multilevel structured description: DIAMII with FORAL," in Data Base Description (Doque, B. C. M., and Nijssen, G. M., eds.), pp. 239-257, 1975.

[SHOS78] Shoshani, A. CABLE: A Language Based on the Entity-relationship Model. Lawrence Berkeley Lab., Berkeley, CA, 1978.

[SIMM73] Simmons, R. F. "Semantic networks: Their computation and use for understanding English sentences," in Computer Models of Thought and Language (Schank, R. C., and Colby, R. F., eds.), pp. 63-113. W. H. Freeman, San Francisco, 1973.

[SMIT77a] Smith, J. M., and Smith, D. C. P. "Database abstractions: Aggregation," Commun. ACM 20, pp. 405-413, 1977.

[SMIT77b] Smith, J. M., and Smith, D. C. P. "Database abstractions: Aggregation and generalization," ACM Trans. Database Syst. 2, pp. 105-133, 1977.

[SPYR80] Spyratos, N. "Translation structures of relational views," Proc. 6th Int. Conf. Very Large Data Bases, pp. 411-416, 1980.

[STIE80] Stiefel, M. "Surveying front end processors," Mini-micro Systems. 13, pp. 129-137, 1980.

[STON75] Stonebraker, M. R. "Implementation of integrity constraints and views by query modification," Proc. ACM SIGMOD, pp. 65-78, 1975.

[STON76] Stonebraker, M. R., Wong, E., and Kreps, P. "The design and implementation of INGRES," ACM Trans. Database Syst. 1, pp. 189-222, 1976.

[STON81] Stonebraker, M. "Operating system support for database management," Commun. ACM. 24, pp. 412-418, 1981.

[SU79] Su, S. Y. W., and Lo, D. H. "A semantic association model for conceptual database design," Proc. E-R Conference, pp. 147-167, 1979.

[SU81] Su, S., Lam, H., and Lo, D. H. "Transformations of data traversals and operations in application programs to account for semantic changes of databases," ACM Trans. Database Syst. 6, pp. 255-294, 1981.

[SUND74] Sundgren, B. "Conceptual foundation of the infological approach to data bases," in Data Base Management (Klimbie, J. W., and Koffeman, K. L., eds.), pp. 61-96. North-Holland, Amsterdam, 1974.

[TAYL76] Taylor, R. W., and Frank, R. L. "CODASYL data-base management systems," ACM Comput. Surv. 8, pp. 67-103, 1976.

[TOME81] Tomek, I. Introduction to Computer Organization. Computer Science Press, Rockville, MD, 1981.

[TSIC76] Tsichritzis, D. C., and Lochovsky, F. H. "Hierarchical data-base management: A survey," ACM Comput. Surv. 8, pp. 67-103, 1976.

[TSIC77] Tsichritzis, D. C., and Lochovsky, F. H. Data Base Management Systems. Academic Press, New York, 1977.

[TSIC78] Tsichritzis, D. C., and Lochovsky, F. H. "Designing the data base," Datamation 24(8), pp. 147-151, 1978.

[TSIC82] Tsichritzis, D. C., and Lochovsky, F. H. Data Models. Prentice-Hall, Englewood Cliffs, NJ, 1982.

[ULLM80] Ullman, J. D. Principles of Database Systems. Computer Science Press, Potomac, MD, 1980.

[VAND81] van de Riet, R. P. et al. "High-level programming features for improving the efficiency of a relational database system," ACM Trans. Database Syst. 6, pp. 464-485, 1981.

[VANG78] van Gigch, J. P. Applied General Systems Theory. Harper and Row, New York, NY, 1978.

[VASS79] Vassiliou, Y. "Null values in data base management: A denotational semantics approach," Proc. ACM SIGMOD, 1979.

[WALT78] Waltz, D. L. "An English language question answering system for a large relational database," Commun. ACM. 21, pp. 526-538, 1978.

[WARN81] Warnier, J. Logical Construction of Systems. Van Nostrand Reinhold, New York, NY, 1981.

[WEIN80] Weinberg, V. Structured Analysis. Prentice-Hall, Englewood Cliffs, NJ, 1980.

[WELD81] Weldon J. -L. Data Base Administration. Plenum Press, New York, 1981.

[WIED77] Wiederhold, G. Database Design. McGraw-Hill, New York, 1977.

[WONG77] Wong, H. K. T., and Mylopoulos, J. "Two views of data semantics: A survey of data models in artificial intelligence and database management," INFOR 15, pp. 344-383, 1977.

[WONGE76] Wong, E., and Youssefi, K. A. A. "Decomposition -- A strategy for query processing," ACM Trans. Database Syst. 1, pp. 223-241, 1976.

[YOUR79] Yourdon, E., and Constantine, L. L. Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design. Prentice-Hall, Englewood Cliffs, NJ, 1979.

[ZANI79] Zaniolo, C. "Design of relational views over network schemas," Proc. ACM SIGMOD, pp. 179-190, 1979.

[ZANI81] Zaniolo, C., and Melkanoff, M. A. "On the design of relational database schemata," ACM Trans. Database Syst. 6, pp. 1-47, 1981.

GLOSSARY OF ACRONYMS

AI: Artificial Intelligence

BNA: Block Numbering Area (geographic area)

BSDM: Basic Semantic Data Model (modeling formalism)

CABLE: ChAin Based LanguagE (data definition/ manipulation language)

CBD: Central Business District (geographic area)

CODASYL-DBTG: COnference on DAta SYstems Languages-DataBase Task Group

DBA: DataBase Administrator

DBMS: DataBase Management System

DDL: Data Definition Language

DM: Database Management (non-AI sphere)

DMF: Data Modeling Formalism

DML: Data Manipulation Language

E-R: Entity-Relationship (modeling formalism)

ERD: Entity-Relationship Diagram

GIS: Geographic Information System

MCD: Minor Civil Division (geographic area)

MIS: Management Information System

MRPPS: Maryland Refutation Proof Procedure System (AI database system)

PL: Programming Language

PL-94: Public Law 94 (census data file)

RM/T: Relational Model/Tasmania (modeling formalism)

SAM: Semantic Association Model (modeling formalism)

SDM: Semantic Database Model (modeling formalism)

SIR: Scientific Information Retrieval (DBMS; version 2 implied)

SMSA: Standard Metropolitan Statistical Area (geographic area)

STF: Summary Tape File (census data file series)

TORUS: TORonto Understanding System (AI database System)

VLDB: Very Large DataBase

wff: well formed formula (predicate calculus) [sic]