The SET CONCEPTUAL MODEL

and the

DOMAIN GRAPH METHOD of TABLE DESIGN

Paul C. Gilmore

Technical Report 87-7
March 1987

ABSTRACT: A purely set-based conceptual model SET is described along with a specification/query language DEFINE. SET is intended for modelling all phases of database design and data processing. The model for an enterprise is its set schema, consisting of all the sets that are declared for it.

The domain graph method of table design translates the set schema for an enterprise into a table schema in which each table is a defined user view declared as a set in DEFINE. But for one initial step, the method can be fully automated. The method makes no use of normalization.

Two kinds of integrity constraints are supported in the SET model. Although these constraints take a simple form in the set schema for an enterprise, they can translate into referential integrity constraints for the table schema of a complexity not previously considered.

The simplicity of the constraints supported in the SET model, together with the simplicity of the domain graph table design method, suggests that a conceptual view of an enterprise provided by the SET model is superior to the lower level data presentation view provided by the relational model.

The research reported on in this paper has been supported by grants of the Natural Sciences and Engineering Research Council of Canada

1.Introduction

1.1. Why Another Model?

To master the complexity of the information processed by a typical enterprise, it is necessary to present abstract descriptions, called data models, or in their more abstract form, conceptual models of the information [Knt78, BMS84]. The earliest models, the hierarchical and network, are low level models in the sense that they are abstract descriptions of implemented data structures that store the required data. The relational model is a higher level model in that it provides abstract descriptions of the data as it is presented to the users in the form of tables. The relational model has freed users from most implementation concerns, but it forces users to be concerned with the presentation of data at the earliest stages of database design when a higher level conceptual view of information needs is more appropriate. Consequences of this insufficiently abstract view of information are evident in the unnecessarily complex normalization methods of table schema design used to ensure the absence of update anomalies, and also in the inability of the relational model to deal with referential integrity in a uniformly simple fashion.

The entity-relationship (ER) model has found a way to avoid both the excessive implementation concerns of the hierarchical and network models, and the restrictive presentation concerns of the relational model.[Chen76,77] The strength of the ER model is that it is neither implementation nor presentation oriented, but rather conceptually oriented. The weakness of the ER model, on the other hand, is that it lacks a sound foundation upon which a management system might be based. The primary motivation for the development of the purely set-based data and conceptual model SET and its specification/query language DEFINE has been to provide such a foundation. There are several reasons why this is necessary:

1. For the unified view of data proposed in [Chen76] to be fully achieved, a database is needed that is capable of recording a high level conceptual model of an enterprise and at the same time of providing the tables for a relational database schema as a defined user view in its specification/query language.

Such a management system will avoid the hand translation process that intervenes between the high level and the user table view of an enterprise [Chen77, HaMc81, Rock81, Knt83a, Chen85, TYF86]. As the enterprise's information needs evolve, such a hand translation will inevitably result in either the conceptual model or the tables being out of date, or very likely the conceptual model, with all its valuable high level information, just being discarded. The SET model integrates the two stages. In the first stage, sets are declared and recorded in a set schema that is a conceptual model for the enterprise; the tables needed for the second stage are just user views of the enterprise defined within the specification/query language DEFINE. As the enterprise's information needs evolve, the model is updated by the declaration of new sets, or by the removal of sets from the set schema. The users' tables are then either automatically updated or if necessary redefined. As illustrated in section 3, much of the design and definition of the tables can be automated.

In as much as the SET model is object-oriented in the sense of [Ditt86], it contributes to the requirement for a high quality database design method called for there.

Queries for a relational database supported by a management system based on the SET model can be posed in DEFINE, or in a language for a relational database schema. The latter would then be translated into DEFINE queries.

2. The modelling process used in ER modelling requires a greater discipline than is now possible.

A tentative beginning is provided in section 2.11 to removing some of the art from conceptual modelling, and in providing a basis for some of the decisions that must be made while modelling. Several principles are stated that should be used in designing conceptual models. These principles cannot be understood without a rigorous foundation such as provided by the SET model.

3. A provably sound foundation is needed for databases that can reference and describe themselves.

A conceptual model for an enterprise requires the capture of large amounts of what is called meta-data that records information about the common data processed in the enterprise's daily activities [Lyke86, DKM86]. As that information is gathered, the meta-data expressing it must be recorded so that it can be queried and updated like any other data. Traditionally data dictionaries have been used for this purpose but they beg the question "where is the meta-data for a data dictionary to be stored?". To avoid an infinite sequence of data dictionaries it is necessary to begin with a database capable of recording information about itself. Although the need for such self-describing databases has been recognized before [LyKe86, Mark85], an awareness of a concern that they raise has gone unnoticed: If such a base is not to fall prey to the paradoxes and contradictions of set theory that shook the foundations of mathematics early in this century, then it must be provided with a provably sound foundation. The necessity for this has been demonstrated again more recently; the semantic networks described in [Sowa84] are subject to the same contradictions as naive set theory [Blac85].

4. A fully unified model of an enterprise is needed that at the same time can give a conceptual view of the enterprise, a user's view of data as it is presented, a data administrator's view of data as it is stored, and a programmer's view of the processing of the data.

The construction of such a unified model of an enterprise is ambitious, but is nevertheless necessary. For without such a unified model there will always remain the need for moving from one model to another when dealing with different aspects of information and its processing, with the probable consequence of the different models becoming inconsistent with one another. The need for such a unified model, raised in [Knt78] and repeated in [LyKe86], is argued forcefully in [TrLo87].

5. Sound foundations are needed for knowledge base systems capable of dealing with incomplete information.

Such systems require distinctions that can only be made precise in a formally described model. For example, it is necessary to distinguish between sets with intensions that can only be understood by humans, and sets with intensions expressed in a language like DEFINE that can be understood by both humans and machines. It is also necessary to distinguish between an internal surrogate for an entity maintained by a management system and a character string that is an identifier for the entity, in order to deal adequately with identity [Knt78, Codd79, KhCo86].

1.2. Summary

The concept of a set or class is one of the most fundamental of mathematics and has been incorporated into scientific and natural languages, as well as used extensively in database theory and practice. It is explicitly used in the relational and entity-relationship models, but also in the entity set [SAAF73], semantic [HaMc81], and functional data models [Ship81, LyKe86], in the system TAXIS [MyWo80], in the techniques described in [FuNe86], and implicitly used in the network data model [TaFr76]. But the concept of set used is an intuitive one and is combined with other related but independent concepts. The SET model, on the other hand, uses set and ordered pair as its only fundamental concepts, with the mathematical foundation for these concepts being provided by the provably consistent set theories of [Gil86a], while other needed concepts are defined in terms of these.

The purpose of this paper is to provide an introduction to a simple form of the SET model, and demonstrate that it supports a unified process of conceptual and presentation modelling called for in (1). A secondary purpose is to demonstrate that it also can contribute to (2). That it can contribute to the remaining three demands, and that DEFINE can be used as a query language, is demonstrated in [Gil87].

In section 2, the elementary features of the SET model and the language DEFINE are introduced through an exercise in conceptual modelling for an enterprise called Simple University, a university so simplified that it does not have any students. The exercise is designed to illustrate how the integrity constraints supported by the model can be used to formalize semantic information about an enterprise.

In section 3 a table design method is described and a table schema for Simple University obtained. Because the set model for Simple University and the presentation model in the form of the table schema are both described in the same language, and because the tables are defined user views, it is possible to prove that the tables correctly record all required data; that is, they are provably free from anomalies. This justification of the method is stated as a theorem in 3.6, the proof of which is postponed to [Gil87]. The method, unlike [TYF86], makes no use of normalization, although it is related to the synthetic method of [Bern76], and the method of [Knt83a].

Using the Simple University example, the integrity constraints supported in the SET model are compared in section 4 with those customarily supported in the relational and network models. The conclusions of the paper are presented in section 5.

Of the cited papers [LyKe86] comes closest to presenting a model with the same intended scope as SET. There are several differences that can be noted between the IRIS model briefly presented there and SET. One superficial distinction is that IRIS is based on functions, while SET is based on sets. But the types of IRIS are sets, and functions are admitted as types. A more profound difference is in the management of declarations. The simple form of the model demands a discipline in the declarations of sets that is absent from IRIS. This may be important to conceptual modelling and to the design of databases.

A detailed description of the syntax and semantics of DEFINE is not given in the paper, but is only introduced as it is needed. It is assumed that the reader is familiar with the semantics for the boolean operators **and**, **or**, and **not**, and for the boolean quantifiers [**For some** ...] and [**For all** ...]. However, the demands on prior knowledge of mathematical logic are minimal throughout the paper.

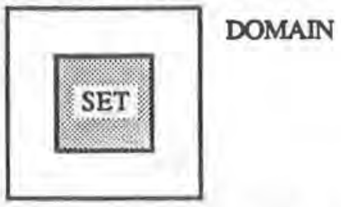
1.3. Acknowledgements

Frege's invention of the predicate logic provided a new standard of precision for mathematical theories that logicians have been exploiting for nearly a century. A demand for a computer implementation of a theory can contribute not only to that standard, but also to the usefulness of the theory. The hard-nosed views of Roderick Morrison in particular, but also of Brian Mullen and of students both undergraduate and graduate, including Michael Black, Julie Abrahamson, Georgis Tsikinis and Michael Kreykenbohm, have contributed to the SET model.

2. Elementary Features of the SET Model

A distinctly computer science view of sets is taken in the SET model. No set can be presumed to preexist; each required set must be explicitly declared. All the declared sets of an enterprise form a **set schema** for the enterprise. A set has an **intension** and an **extension**. The intension of a set is the property that an object must possess to be a member of the set; the extension of a set is the collection of objects satisfying the intension of the set. The intension of a set is to be thought of as time invariant, while the extension of a set changes as circumstances change. The intension of a set may be expressed in a natural language statement intended for human understanding only (the **base** sets), or may be expressed in a language like DEFINE that can be understood by both humans and machines (the **defined** sets). The extension of a set is drawn from its **domain** as illustrated in figure 2.1.

FIGURE 2.1



The domain of a set S is the extension of the cartesian product of one or more previously declared sets. Each set DS occuring in the cartesian product is an **immediate domain predecessor** of S of **multiplicity** the number of times DS occurs in the product. To avoid an infinite regression of domains, it is necessary to assume that some sets are declared without domains; these are the **primitive** sets.

2.1 Simple University

To illustrate these and other elementary features of the model their application to the following example will be demonstrated: At Simple University (SU) there are departments, each identified by its name. Each employee of SU is a member of exactly one department, and is identified by a number. Each course taught at SU is identified by a course number and the department responsible for it. An academic department is one responsible for some courses. An instructor is a member of an academic department who is competent to teach some of the courses of the department to which he/she belongs; an instructor is currently teaching some of the courses he/she is competent to teach. When taught, a course is taught by exactly one instructor. Exactly one instructor of an academic department is manager of the department. Exactly one member of a nonacademic department is manager of the department. Although in a more realistic example employees, departments, and courses, would have attributes declared for them, they will be largely ignored so as to concentrate on more important features of conceptual modelling.

The set of departments of SU is a primitive base set. This set can be declared as follows:
D={ D || the current departments of SU }
The name D of the set appears to the left of =. The fact that D is a primitive set is indicated by the second occurrence of D, where normally the domain of a declared set is recorded. The two vertical bars separate the domain declaration from a comment that expresses the intension of D. Since only humans can determine whether an entity satisfies the intension of D, it is a base set. In declarations of nonprimitive base sets and of defined sets, a machine interpretable expression appears between the two vertical bars.

2.2 Internal Surrogates and Identifiers

The result of the declaration of a primitive base set for a management system is the preparation of a file where internal surrogates of members of the set can be recorded. When a user of the system indicates that a new member is to be added to the set, the system generates a new distinct internal surrogate for the entity. Thus at any time the internal surrogates recorded by the system as members of the set should be in one-to-one correspondance with the entities that users perceive as members of the set.

The internal surrogate of a department must be distinguished from an identifier of the department. The former need not be known to users of the system, while the latter, as stated in the description of SU, is its name. The names of departments are particular strings of characters that can be written and read by both humans and machines. Such strings are the means by which humans and machines communicate with each other. For example, if 'ENGLISH' is the name of one of the departments of SU, then the management system will have been instructed to associate the string with the internal surrogate of a member of the set D that was created when the English Department was added to the set. By using the string 'ENGLISH' in an appropriate way, a user can convey information to the management system about the department represented by the internal surrogate, and subsequently ask for information about the department.

2.4. The Kernel Schema and Value Sets

The strings that are names of departments are members of a set STR of all strings that is supported by the management system. In programming language terminology, it is a data type. Another such data type is INT, the set of integers. Each of these is a primitive defined set. They are primitive sets since there is not a previously declared set from which their extensions can be drawn, and they are defined since the management system can determine whether an entity is or is not a member of it.

The declaration of STR is:
STR={ x:STR | x system defined | the set of strings of characters }.
The assertion 'x:STR' in the declaration expresses two things: First that the domain of STR is STR, and second that the variable 'x' is declared to be a member of STR. This is typical of the declaration of a defined set. The assertion 'x system defined' expresses that the membership of STR is determined by the system; it is assumed to be a primitive assertion of DEFINE that can be read and understood by both humans and machines. Finally the phrase 'the set of strings of characters' is an informal comment describing the membership of STR in human understandable terms only; such a comment may be added to the declaration of any defined set, but must not be confused with the intension of the set.

The declaration of INT is:
INT={ x:INT | x system defined | the set of integers }.
This declaration, along with that of STR, are among those declarations of sets needed by the system managing set schemas; they form what is called the **kernel schema** of the system. Users of the system cannot change the kernel schema in any way, although the declarations must, of course, be available to users. Kernel schemas are discussed at greater length in [Gil87].

It is possible, and at times useful, to regard the members of INT and STR as entities with internal surrogates and surface identifiers, just like the members of base sets, even though they are defined sets [Morr]. In this paper, however, the simpler view of these sets as sets of strings will be maintained.

Nonprimitive sets must also be declared in the kernel schema:
≤={ x:INT, y:INT | <x,y> system defined | x is less than or equal to y }.
Since the domain of ≤ is the cartesian product of previously declared sets, ≤ is a nonprimitive set; INT is an immediate domain predecessor of ≤ of multiplicity 2. The declaration of ≤ makes it possible to express that an integer x is less than or equal to an integer y by the usual infix notation 'x ≤ y', although the standard form of the infix notation in DEFINE is 'x:≤:y', the colons being used to separate the two arguments from the name of the set.

Another example of a nonprimitive set declared in the kernel schema is:
L={ x:STR, y:INT | <x,y> system defined | x is a string with length y }.
L is a nonprimitive defined set with extension those pairs <x,y> admitted by the system. The declaration of L makes it possible to express that a string x has length an integer y with the assertion 'x:L:y'. The same fact can be also expressed with the assertion '<x,y>:L'. The two assertions 'x:L:y' and '<x,y>:L' have exactly the same meaning, but the infix form will be seen to be more convenient at times.

L is usually thought of as a function; that is, it takes an argument x and returns a value y. Expressed in the usual functional notation, y is L(x). The intension of L, when elaborated, describes how the value L(x) is determined from the argument x. But the extension of L is nevertheless a set of pairs <x,y>, where x is a member of STR and y is a member of INT. To avoid syntactic confusions, the usual functional notation will not be used, but rather a notation that emphasizes the fact that functions are just sets of pairs. For example, instead of 'L(x)', the notation '{x:L:}' will be used. The semantics of the notation is provided in the manner of the functional notation of [Gil77].

Not all members of STR are admitted as names of departments, but only those of length at most 10 characters. The subset of such strings is declared as follows:
VN={ x:STR | {x:L:} $\leq$ 10 | a value set for names of departments }.
The assertion {x:L:} $\leq$ 10 expresses that for x to be a member of VN, it must have length not exceeding 10.

The comment in VN refers to it as a **value set**. These sets are defined recursively as follows: Each of the primitive defined sets STR and INT is a value set, the cartesian product of value sets is a value set, and any defined set with domain a value set is a value set. The sets $\leq$, L, and VN are all value sets by this definition, but unlike the first two sets, VN is not declared in the kernel schema, but is a user declared set. Generally the particular form of such sets is one of the least important considerations in the design of a set schema. Therefore good design practice dictates that the statement of the intension of value sets within DEFINE be postponed as long as possible.[Knt83a]

2.5. Degrees of Base Associations

With each member of D, a unique member of VN must be associated. The required declaration is:
DN={ D, VN | <1,1>, <0,1> | a unique name from VN is associated with each department }.
The domain of DN is the cartesian product of its immediate domain predecessors D and VN listed in that order in the declaration. The intension of DN follows the second of the two vertical bars in the form of a comment. Since the comment cannot be interpreted by machine, DN is a base set: The names to be given to the departments of SU are determined by humans. The machine interpretable expression between the two vertical bars is a **degree** declaration. It consists of two pairs of degrees, with the first pair relating to D and the second pair to VN, which is in the order specified in the domain declaration.

The first element of a pair of degrees is the **lower degree** and may be 0 or 1, while the second is the **upper degree** and may be 1 or *. The lower degree of the first pair <1,1> of degrees states that for every member of D there is at least one member of VN associated with it; that is, each department must have a name associated with it. The upper degree of the first pair states that with a member of D no more than one member of VN can be associated; that is, each department must have at most one name associated with it. The constraints expressed by the first pair <1,1> of degrees ensures, therefore, that each department has a single name associated with it.

The lower degree 0 of the second pair <0,1> of degrees expresses that not every member of VN need be associated with a member of D; that is, not every member of VN is necessarily the name of a department. The upper degree 1 of the second pair expresses that a member of VN can be associated with at most one member of D; that is, the same name is never assigned to two departments. The constraints expressed by the second pair <0,1> of degrees ensures, therefore, that if a member of VN is used as a name, then it is used as the name of a single department. The constraints expressed by the two pairs of degrees <1,1> and <0,1> ensures that each department has a single name that is unique to the department.

The need for degrees has been widely recognized. In [Knt83a, LyKe86] they are called respectively the least and maximum participation, and in [LeSa83] the minimum and maximum cardinalities. In [LyKe86] the upper degree * is denoted by m. In [TYF86] the upper degrees are referred to as the cardinalities of the connectivity of a relationship, while the lower degrees are described as optional or mandatory connectivity. They have been called degrees in the SET model because they can be regarded as lower and upper bounds on the degrees of nodes of bipartite graphs representing the associations. For example, the bipartite graph representing the DN association has one set of nodes representing the members of D and one set of nodes representing the members of VN. The undirected edges of the bipartite graph connect each member of D to the member of VN that is DN associated with it. The degree of a node in a graph is the number of edges that connect to it.

Degrees can be declared for base sets that have any number of immediate domain predecessors. For example, a base set P with domain Q can be declared to have lower degree 0 or 1, 0 meaning that the extension of P is a proper subset of the extension of Q, and 1 meaning that they are extensionally identical. Since there is no need to declare P as a base set if it is to be extensionally identical to Q, only a lower degree of 0 is needed. Also only an upper degree of 1 makes sense, since for each member of P there is exactly one member of Q.

Degrees can also be declared for sets with more than two immediate domain predecessors. For example, if A is declared with base PxQxR, then the lower degree of A on P is the least number, for any p, of pairs <q,r> for which <p,q,r> may be a member of A. The upper degree of A on P is similarly defined.

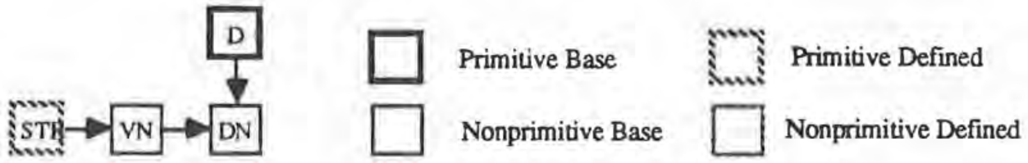
Degrees are declared only on base sets, and are **integrity constraints** expressing restrictions or constraints on the membership of the sets. They should be regarded as part of the intension of the sets; they are a machine readable part of an otherwise machine unreadable intension. The degree constraints, together with the implicit inclusion constraints expressed when the domain of a set is declared, are the only integrity constraints that can be expressed in the SET model.

Degree constraints can provide the foundation for the dependency theories of the relational model and are shown, in section 3, to be fundamental to the design of a table schema that can record data without anomalies. In [Gil87] the satisfiability and maintainance of degree constraints are discussed.

2.6. A Domain Graph

Graphical representations are an important way of providing a user with an overview of declarations that have been made. The diagrams of the ER model and the Bachman diagrams [Bach69] of the network model are examples of such representations, as are illustrations of domain graphs for the SET model, an example of which is given in figure 2.2. It must be stressed, however, that an illustration of a domain graph is not a substitute for set declarations, but only provides an overview of the domain declarations of the declared sets.

FIGURE 2.2



A **domain graph** of a set schema is a directed graph with nodes labelled with declared sets of the schema, at most one node for each set, and with directed edges <nde1, nde2> for which the tail nde1 is labelled with an immediate domain predecessor of the head nde2. In figure 2.2 each of the sets D, STR, VN, and DN labels a node, and the arrows point from the immediate domain predecessor STR of VN to VN, and from the immediate domain predecessors D and VN of DN to DN. The boxes representing the nodes of this simple domain graph are drawn with different lines simply to emphasize the four kinds of sets that have been declared.

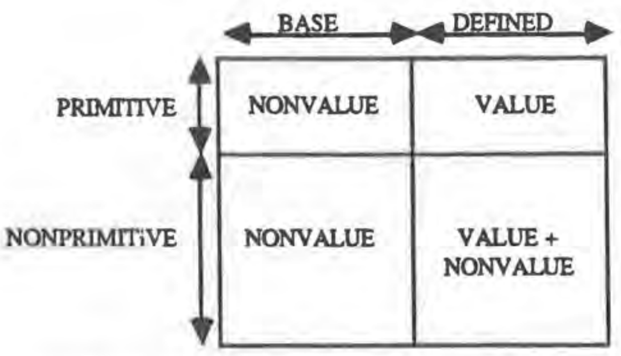
In the domain graph illustrated in figure 2.2, at most a single edge appears between any two nodes. That need not always be so. For example, in a domain graph with nodes labelled with the system declared sets $\leq$ and INT, there would be two edges directed from the node labelled with INT to the node labelled with $\leq$, since the immediate domain predecessor INT of $\leq$ has multiplicity 2.

Since the immediate domain predecessors of a set must be declared previously to the set, a domain graph of a set database schema is necessarily directed acyclic; that is, it is not possible to follow a path from a node back to itself by traversing edges in their specified direction. However, a domain graph may be undirected cyclic; that is, it may be possible to follow a path of edges from a node back to itself if edges are traversed in any direction. The domain graph with nodes labelled with $\leq$ and INT, for example, has an undirected cycle.

2.7. An Ontology of Sets

The declared sets in a set schema are either primitive or nonprimitive, and either base or defined. There are therefore four possible classes of sets. These are illustrated in the figure 2.3.

FIGURE 2.3



D is primitive base and not a value set, STR and INT are primitive defined and are value sets, DN is nonprimitive base and not a value set, and VN, $\leq$, and L are nonprimitive defined and value sets. The diagram indicates that all primitive defined sets are value sets; this is true of the sets declared so far since STR and INT are the only primitive defined sets that are value sets. The diagram illustrates the simplest form of the model that will be used throughout this paper. That form of the model is adequate for declaring the set schemas of ordinary enterprises, but the more general form of the model in which not all primitive defined sets are value sets is needed to declare the kernel schema described in [Morr] and discussed in [Gil87].

The primitive defined sets that are value sets are provided by the system with identifiers. For example, the members of INT can be regarded as strings that identify themselves since they are strings that can be read and written by both humans and machines. To refer to a member of INT, it is only necessary to state it as a string. The members of STR are also such strings; to refer to a member of STR it is only necessary to enclose it in quotes. A primitive base set, on the other hand, must have an identifier declared for it that provides a one-to-one association between its members and a subset of a value set. The set DN is an identifier for the primitive base set D. Nonprimitive sets inherit identifiers from their immediate domain predecessors. For example, if dep is the internal surrogate of the department with name 'ENGLISH', then the pair <dep, ENGLISH> is a member of DN and is identified by the pair <ENGLISH, 'ENGLISH'>, since the first element of the pair identifies dep, and the second element the member of VN that is the name of dep.

2.8. Employees and Courses

The next set to be declared for the SU schema is

E={ E ll the set of current employees }.

Employees are identified by employee numbers.

VE#={ x:INT | $1000 \leq x \leq 9999$ | }

E#={ E, VE# | <1,1>, <0,1> | each employee has a unique employee number }

Each employee is assigned to a single department. The association between employees and departments is declared next:

ED={ E, D | <1,1>, <1,*> | associates each employee with a unique dept }

The second pair of degrees in this case indicates that a department must have at least one member and that it can have any number of members.

A course is identified by a department responsible for it and a course number. Course numbers are selected from a value set:

VC#={ x:INT | $100 \leq x \leq 699$ | }.

A course can therefore be regarded as an association between departments and course numbers:

C={ D, VC# | <0*>, <0*> | a course is identified by a responsible dept and a course # }.

The lower degrees in this case mean that not every department is responsible for courses, and not every member of VC# is a course number. The upper degrees mean that a department can be responsible for any number of courses and that a course number may be the number for any number of courses.

An alternative to the given declaration of C would be to declare it as a primitive base set and then to declare an association IDC that identifies members of C through members of DxVC#. From the IDC association there then could be defined the associations between C and D and between C and VC#. The declaration chosen for C avoids the need for additional declarations, although it does so at the price of some artificiality.

If it is desired that course numbers carry additional meaning, such as the year in which a student is expected to take the course, then that meaning should be expressed as part of the intension of the C association. Such a restraint can of course only be enforced by those who assign numbers to courses.

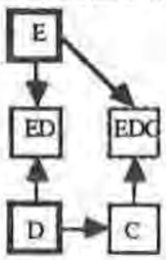
2.9. Defined Sets and Define Predecessors

The next set to be declared requires some explanation. An academic department is one responsible for courses; an instructor is a member of an academic department who is competent to teach some of the courses of his/her department. Let ICC be the association between an instructor and a course the instructor is competent to teach. ICC is clearly a nonprimitive base set and a subset of ExC. However, ICC cannot be just any subset of ExC, for an instructor is restricted to being competent to teach courses only of his/her department. If EDC is the association between an employee and all the courses for which the employee's department is responsible, then clearly ICC has EDC as its domain. EDC can be declared as a nonprimitive defined set; it is the first example of such a set that is not a value set:
EDC={ x:E, <y,z>:C | <x,y>:ED | associates the courses for which an academic department is responsible with a member of the department}.

The declaration of a defined set such as EDC has two formal parts and one optional informal part. The optional informal part is a comment on the intension of the set. The first of the formal parts consists of the **domain assertions**, one or more elementary assertions such as the assertions x:E and <y,z>:C for EDC. These assertions accomplish two purposes. First they declare the domain of the defined set, so their order is significant; for EDC the domain is ExC. Second they declare the range of the variables or tuples or nested tuples of variables that appear in them. For EDC the range of the variable x is declared to be the set E, and the range of the pair of variables <y,z> is the set C. Although the latter declaration has the effect of declaring y to be restricted to D and z restricted to VC#, it is of course not equivalent to the domain assertions y:D and z:VC#. To ensure a proper declaration of the domain to which it is to be bound, a variable can have at most a single occurrence among the domain assertions of the declaration.

The second of the formal parts of the declaration of a defined set is an assertion of DEFINE that expresses the set's intension. For EDC the intension is the assertion <x,y>:ED. The domain assertions together with the intension of the declaration determines the extension of the set. The pairs <x,<y,z>> that are members of EDC are those for which x is a member of E, <y,z> is a member of C, and <x,y> is a member of ED. They are therefore those pairs <x,<y,z>> for which the employee x is assigned to the department y responsible for the course <y,z>. The domain graph illustrated in the figure 2.4 shows how EDC is related to E, D, and ED.

FIGURE 2.4



Notice that no degrees are declared for the defined set EDC. Since the membership of EDC is determined by the system from its intension, the degrees of EDC on E and C must follow from that intension, that is, they can be calculated. Although the calculation of the degrees of defined sets is in general an unsolvable problem, they can be quite simply determined for the defined sets usually declared for the set schema of an enterprise. The degrees for EDC, along with the degrees of other defined sets for the set schema of Simple University, are calculated in section 3.1.

The set ICC can now be properly declared:
ICC={ EDC | <0,*>, <1,*> | some employees of an academic department are competent to teach some of the courses of the department }.
The meaning of the degrees of ICC will be described in 2.11, after some required definitions are given here and in 2.10.

The defined set EDC, like the previously declared defined sets VN and VE#, is of little direct interest. Its primary purpose is to provide a domain for ICC. Consequently a management system is unlikely to keep a list of internal surrogrates of members of EDC, but rather use the definition of EDC to check that proposed members of ICC are members of EDC. EDC is therefore called a **virtual** defined set, as opposed to an **actual** defined set for which the management system is instructed to maintain a list of internal surrogates that are members. In an implementation of the SET model it is expected that a user could declare whether a defined set should be virtual or actual.

The sets E and C are immediate domain predecessors of EDC. The set ED used in the intension of EDC is an **immediate define predecessor** of EDC. Every defined set, apart from the system defined sets appearing in the kernel schema, has one or more immediate define predecessors. An **immediate predecessor** of a set is either an immediate domain predecessor or an immediate define predecessor. The immediate domain predecessors are the only immediate predecessors of a nonprimitive base set, while a nonprimitive defined set such as EDC has immediate predecessors that are immediate domain predecessors and ones that are immediate define predecessors. A **predecessor graph** for a set schema is obtained by adding edges to the domain graph corresponding to the immediate define predecessors of defined sets. Like a domain graph, a predecessor graph must be acyclic because sets must be declared before they can be used in the intension of a declared set. It is sometimes useful to display a predecessor graph in a diagram as has been done with domain graphs, although that is not done in this paper.

A **predecessor** of a set is defined recursively: It is either an immediate predecessor of the set or an immediate predecessor of a predecessor of the set. That one set is a predecessor for a second means that the extension of the second set can be dependent upon the extension of the first. For example, although ICC is a base set, it has a long list of predecessors, namely EDC, E, C, ED, D, VC#, and INT, that can affect its extension.

2.10. Arity Domains, Arity Predecessors, and Arity

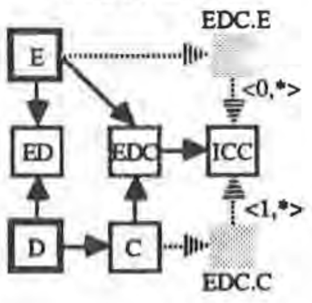
The members of EDC are pairs <x,y>, with x a member of E and y a member of C; it is a **binary** set because it has two immediate domain predecessors. The members of ICC are also pairs although ICC has only the single immediate domain predecessor EDC; but that single predecessor is binary, so ICC is binary also. EDC is the **arity domain** of ICC, defined recursively as follows: The arity domain of a primitive set, or of a set with two or more immediate domain predecessors, is the set itself. The arity domain of a set with a single immediate domain predecessor is the arity domain of that predecessor. Since EDC has two immediate domain predecessors, it is its own arity domain. Since EDC is the only immediate domain predecessor of ICC, the arity domain of EDC is the arity domain of ICC.

An **arity predecessor** of a nonprimitive set is any immediate domain predecessor of the arity domain of the set. E and C are the two arity predecessors of ICC. The **multiplicity** of an arity predecessor is the multiplicity of it for the arity domain. Each of E and C have multiplicity 1 since they occur only once in the cartesian product that is the domain of EDC. The **arity** of a set is 1 if its arity domain is primitive, and is otherwise the sum of the multiplicities of its arity predecessors. The arity of ICC is the sum of the multiplicities of E and C, which is 2. An arity 2 set is said to be binary, and an arity 3 set ternary.

2.11. The Degrees of Base Sets that are not Arity Domains

Consider now the degrees declared for ICC. Since ICC is not its own arity domain, these degrees are declared relative to the projection of EDC, its immediate domain predecessor, on the immediate domain predecessors E and C of EDC. This is illustrated in figure 2.5.

FIGURE 2.5



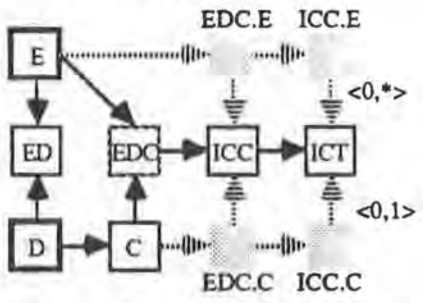
The projections EDC.E and EDC.C are not declared, although they could be if desired. For example, EDC.C could be declared
EDC.C={ y:C | **[For some** x:D] <x,y>:EDC | the projection of EDC on C }.
EDC.E is the set of employees of academic departments and is a proper subset of E, while EDC.C is the set of courses for which departments are responsible and has therefore the same extension as C.

The graph illustrated in figure 2.5 is not a domain graph because the sets EDC.E and EDC.C have not been declared. It is called an **augmented domain graph**.

The degrees <0,*> of ICC on EDC.E mean that not all employees of academic departments are competent to teach courses, and that an employee of such a department may be competent to teach any number of courses. The degrees <1,*> on EDC.C mean that for every course there is an employee of an academic department competent to teach it, and that there may be any number of such employees.

The next set to be declared is ICT:
ICT={ ICC | <0,*>, <0,1> | associates an instructor with the courses he/she is currently teaching }
ICT also has EDC as its arity domain, since its immediate domain predecessor ICC has EDC as its arity domain. The degrees of ICT are therefore declared relative to the projections ICC.E and ICC.C as shown in the augmented domain graph illustrated in the next figure.

FIGURE 2.6



ICC.E is the set of instructors since each member of the set is competent to teach at least one course. ICC.C has again the same extension as C since for every course there is an instructor competent to teach it.

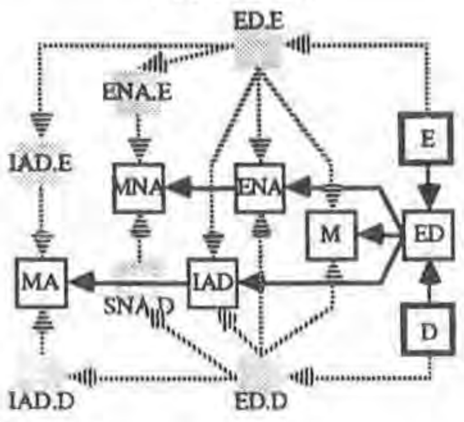
2.12. Managers of Departments

There remains now to declare the sets needed to express the manages associations between departments and employees. Because the manager of an academic department must be an instructor of the department, and the manager of a nonacademic department must be a member of the department, it is necessary to again declare some defined sets before declaring some base sets with the defined sets as domains:
IAD={ <x,y>:ED | **[For some** v:VC#] (<y,v>:C **and** <x,<y,v>>:ICC) | associates an instructor with his/her academic dept }
MA={ IAD | <0,1>, <1,1> | an instructor of an academic dept manages it }

ENA={ <x,y>:ED | **not [For some** v:VC#] <y,v>:C | associates employees of nonacademic depts with their depts }
MNA={ ENA | <0,1>, <1,1> | an employee of a nonacademic department manages it }
M={ z:ED | z:MA **or** z:MNA | associates manager of a dept with the dept }

The augmented domain graph of the next figure illustrates the relationships between these declared sets and the projections that are not declared.

FIGURE 2.7



The immediate predecessors of M are the sets ED, MA, and MNA, but among its predecessors is the set ICC. This means that changes in the extension of the base set ICC could result in changes to the extension of M.

2.13. A Set Schema for Simple University

A summary of the declarations of all the sets of the schema for Simple University is provided in the next figure. As noted before, a more realistic example would have attributes declared for some of the declared sets, but they are unnecessary for the purposes of this paper.

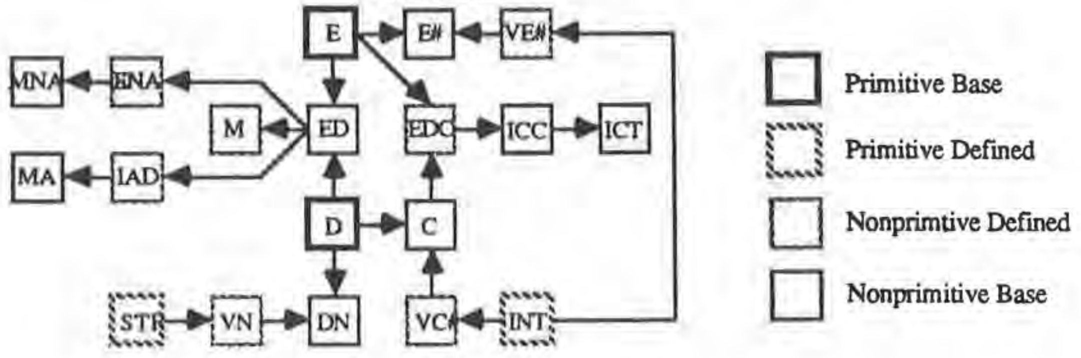
FIGURE 2.8

NAME	DOMAIN	INTENSION / COMMENT
STR	x:STR	x system defined | the integers
INT	x:INT	x system defined | the strings of characters
≤	x:INT, y:INT	x system defined | x is less than or equal to y
L	x:STR, y:INT	<x,y> system defined | y is the length of x
D	D	| a current department
VN	v:STR	{x:LNG:} ≤ 10 | a value set for names of departments
N	D, VN	<1,1>, <0,1> | identifies a dept by a name
E	E	| current employees
VE#	u:INT	1000 ≤ u ≤ 9999 | a value set for employee numbers
E#	E, VE#	<1,1>, <0,1> | identifies an employee by a number
ED	E, D	<1,1>, <1,*> | associates emp with a single dept
VC#	w:INT	100 ≤ w ≤ 699 | a value set for course numbers
C	D, VC#	<0,*>, <0,*> | identifies course by dept and course #
EDC	x:E, <y,z>:C	<x,y>:ED | an employee's dept's courses
ICC	EDC	<0,*>, <1,*> | instructor competent to teach courses
ICT	ICC	<0,*>, <0,1> | instructor currently teaching courses
IAD	<x,y>:ED	**[For some** v:VC#] (<y,v>:C **and** <x,<y,v>>:ICC) | associates an instructor with his/her academic dept
MA	IAD	<0,1>, <1,1> | an instructor of an academic dept manages it

ENA	<x,y>:ED	**not** [**For some** v:VC#] <y,v>:C | associates emp of nonacademic depts with their depts
MNA	ENA	<0,1>, <1,1> | an emp of a nonacademic department manages it
M	z:ED	z:MA **or** z:MNA | associates manager of a dept with the dept

The first four declarations in this schema are part of the kernel schema. A domain graph for the schema of SU is illustrated in the next figure.

FIGURE 2.9



2.14 Principles of Set Modelling

The existence of a precise model to support the earliest stage of conceptual modelling permits the statement of principles that can be used to guide that modelling. Principles of design that have been followed in the modelling of Simple University are:

+ A primitive base set should have as members only those entities of interest to the enterprise, but should be as large as possible consistent with the provision of a simple identifier.
+ A set that can be declared to be defined should not be declared to be base.
+ Machine maintainable constraints on the membership of a base set should be expressed in the intension of a defined set that is the domain of the base set.

The primitive base set E has been declared so as to include all employees of SU, and D has been declared so as to include all departments. There are subsets of each of these sets that are important to SU, for example the instructors which are a subset of E, and the academic departments which are a subset of D. But these subsets should not be declared as primitive base since the larger sets can be declared just as easily. However, a primitive base set with extension employees and departments has not been declared because a natural identifier for such a set would be difficult to provide.

Although the set of academic departments has not had to be explicitly declared, it could be declared as follows:
AD={ x:D | [**For some** y:VC#] <x,y>:C | the academic departments }.
It is clearly a defined set, not a base set, so that its extension can be maintained by the system. To declare AD as a base set BAD would be a serious mistake. For the extension of BAD would then have to be maintained by humans; in order to maintain the integrity of the model of SU it would be necessary for them to ensure that the extension of BAD was at all times the same as the extension of AD, an unnecessary task that can be eliminated by declaring AD rather than BAD. In [TYF86] a weak form of defined set is called a redundant relationship and the principle is recognized that redundant relationships should be eliminated.

Instructors of SU are never declared to be competent in courses of departments other than their own. The base set ICC could have been declared to have ExC as its domain, and the constraint could be maintained by users of the system. But it is better to have the system maintain the constraint to ensure that it is not violated. To accomplish this EDC was declared as a defined set and ICC declared as a base set with EDC as its domain.

Another principle that has been followed in the modelling of SU is the following:

+ A set that can be declared to be nonprimitive should not be declared to be primitive.

The set C has been declared as nonprimitive, although it could be declared as primitive. Although some artificiality results from this declaration, it is compensated for by the resulting simplification in the set schema. Nevertheless this principle is one that should be followed with care. Carried to the extreme, the principle could result in the set D being declared as a base set with domain VN, and E being declared as a base set with domain VE#.

Another principle expressed in [Gil86b], but not relevant for the SU example, concerns the declaration of sets of arity greater than two. No such base set should be declared with upper degree 1 on any of its immediate predecessors, since such a degree indicates that the set can be naturally defined in terms of sets of lesser arity. [TYF86] also recognizes the need for such a principle.

3. The Domain Graph Method of Table Design

Tables are commonly used presentation data structures. Given a set schema, it is often desirable to declare a table schema capable of correctly recording the extensions of declared sets of the set schema. In this section a method for designing such a table schema is described. The resulting table schema is a user view of the set schema in the sense that each table in the schema is declared as a defined set and becomes an additional declared set of the set schema. The method is best described in terms of operations on the augmented domain graph of the set schema. For this reason it is called the domain graph method of table design.

A summary of the steps in the method are:

1. Each edge of the augmented domain graph of the set schema is labelled with the lower and upper degrees that have been declared or calculated for it. The calculation of the degrees for defined sets requires human intervention.
2. Subgraphs of the augmented domain graph are determined by selecting edges that have been labelled with a lower degree 1, or with the degrees <1,1>. Which subgraphs to select requires design decisions as to which sets should have their extensions recorded, and what kinds of tables are acceptable. However, the latter decision can be automated if one kind of table is always acceptable. The resulting subgraphs are simplified by eliminating all nodes labelled with undeclared sets, and by replacing directed paths through such nodes with a single edge connecting nodes labelled with declared sets.
3. Each undirected cycle of a subgraph determined in 2 is broken by removing an edge of it with tail a bottom node of the cycle. The result of this step is a forest of trees.
4. Each tree obtained in 3 is extended with new nodes and edges to form its identifier extension. In the tree that is the identifer extension of a given tree, every set labelling a node in the tree has an identifier labelling a node of the tree.
5. From the identifier extension of each tree obtained in 4, a declaration of a table as a defined set is constructed.

Each set, that was selected in 2 to have its extension recorded, will label a node of exactly one of the subtrees obtained in that step. Each subtree selected will result in a single table of the table schema obtained in 5, so that the number of subtrees selected is the number of tables that will appear in the table schema. That number is the minimum possible consistent with the decision made in 2 to keep only edges of lower degree 1, or to keep only edges of degrees <1,1>, if the subgraphs selected in 2 are maximal.

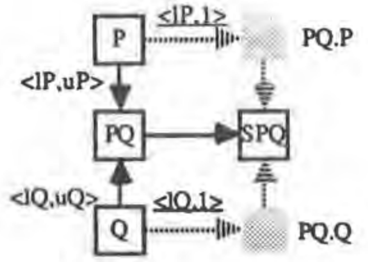
The subsections 3.1 through 3.5 are devoted to these five steps. In 3.6 a justification for the method is stated and proved. It is shown that the extension of each set labelling a node of a subtree obtained in step 2 is correctly recorded through its identifers in the table constructed for the subtree. Finally in 3.7 motivations for the design decision of step 2 are discussed.

3.1. Degrees for all the Edges of an Augmented Domain Graph

Consider the edges of an augmented domain graph. Each edge with head a node labelled with a base set, has a tail that is a node labelled with an immediate domain predecessor of the set. Each such edge can therefore be assumed to have been labelled with a pair of degrees, since the degrees for a base set of arity one are always assumed to be <0,1>, while those of base sets of arity greater than one are always declared. Edges of the augmented domain graph remaining to be labelled are therefore of two kinds, those that are directed to nodes labelled with undeclared but implicitly used sets, and those that are directed to nodes labelled with defined sets.

Consider first those edges that are directed to nodes labelled with undeclared but implicitly used sets. In figure 3.1 a typical case for such sets is illustrated.

FIGURE 3.1

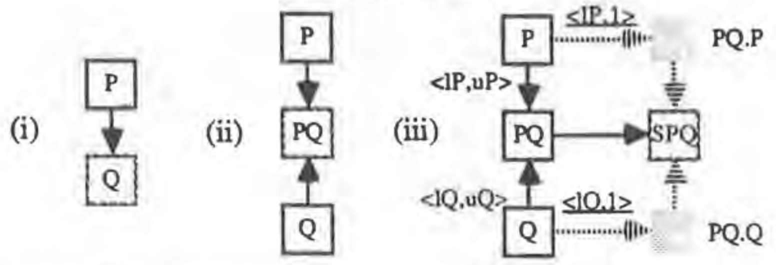


The sets P and Q may have been declared as either base or defined, or they may themselves be undeclared sets. In either case the sets PQ and SPQ will have been declared as either base or defined, with PQ the domain of SPQ. For example, in figure 2.6, P, Q, PQ, and SPQ could be respectively E, C, EDC, and ICC, or they could be respectively EDC.E, EDC.C, ICC, and ICT. In figure 2.7, they could be respectively E, D, ED and any one of IAD, ENA, or M. The degrees of PQ on P have been declared or calculated to be <lP,uP>, and those on Q to be <lQ,uQ>.

Degrees for the edge from P to the undeclared set PQ.P, and the edge from Q to the undeclared set PQ.Q, must be calculated. The upper degrees are both necessarily 1 since PQ.P is a subset of P, and PQ.Q a subset of Q. The lower degree for the edge from P to PQ.Q is necessarily lP, the lower degree of PQ on P, and the lower degree for the edge from Q to PQ.Q is necessarily lQ. In the diagram the calculated degrees have been underlined.

Consider next those edges that are directed to nodes labelled with defined sets. In figure 3.2 three typical cases are illustrated.

FIGURE 3.2



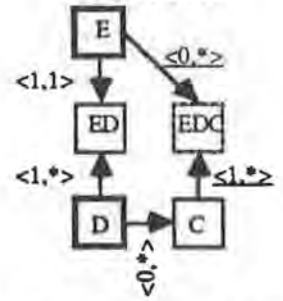
In case (i) Q is defined with domain P, in case (ii) PQ is defined with domain PxQ, and in case (iii) SPQ is defined with domain PQ. The illustration in case (iii) is a repetition of figure 3.1. An example of case (i) appears in figure 2.2 where P is STR and Q is VN. An example of case (ii) appears in figure 2.4 where P, Q, and PQ are respectively E, C, and EDC. Three examples of case (iii) appear in figure 2.7; in each case P, Q, and PQ are respectively E, D, and ED, while SPQ is IAD, ENA, or M.

It is not possible in general to calculate the degrees of a defined set, since given a sufficiently rich language for stating the intensions of defined sets, it is possible to have the extension of a defined set express solutions to unsolvable problems, such as the halting problem for Turing machines. However, for the defined sets usually declared for a set schema of an enterprise, the degrees can be calculated. This is the case for the degrees of all the defined sets in the set schema for Simple University.

VN, VE#, and VC# are the only defined sets of the set schema that fall under case (i). The degrees for each of these on their domains are clearly <0,1> since each is a proper subset of its domain. Indeed, this will in general always be the case for (i), since there is no need to declare Q if it always has the same extension as P.

EDC is the only example to fall under case (ii). To see how the degrees of EDC are calculated consider figure 3.3, in which is illustrated the domain graph of figure 2.4 with degrees labelling its edges.

FIGURE 3.3



The calculated degrees for EDC are underlined, while those that have been declared are not. Consider the undirected path from E to C via ED and D. There are two edges on this path directed in the direction of the path, the edge from E to ED, and the edge from D to C. The product of the lower degrees 1 and 0 respectively of these two edges is 0. Therefore the lower degree of EDC on E is 0. The product of the upper degrees 1 and * is *. Therefore the upper degree of EDC on C is *. Now consider the path from C to E. Since each member of C is a pair with first element a member of D, for every C there is a member of D. Further, the lower degree of the edge from D to ED is 1. Therefore the lower degree of EDC on C is 1. Every course has a single responsible department, but the upper degree of the edge from D to ED is *. Therefore the upper degree of EDC on E is *.

Consider now the case (iii) for the defined sets IAD, ENA, and M; it is helpful to refer to figure 2.7. Their degrees on the undeclared sets ED.E and ED.D can be calculated. First note that ED.E has the same extension as E and that ED.D has the same extension as D, since the degrees of ED are <1,1> on E and <1,*> on D. Therefore the degrees of IAD and ENA on ED.E are <0,1> and <0,*> on ED.D, since not every employee is an instructor and not every department is academic.

The degrees of M are more difficult to calculate. First note that the union of IAD and ENA is a subset of ED, and that M is a subset of this union. Therefore the degrees of M on ED.E must be <0,1>. However, since the degrees of MA on IAD.D and of MNA on ENA.D are <1,1>, and a department is either academic or not, the degrees of M on ED.D are <1,1> also.

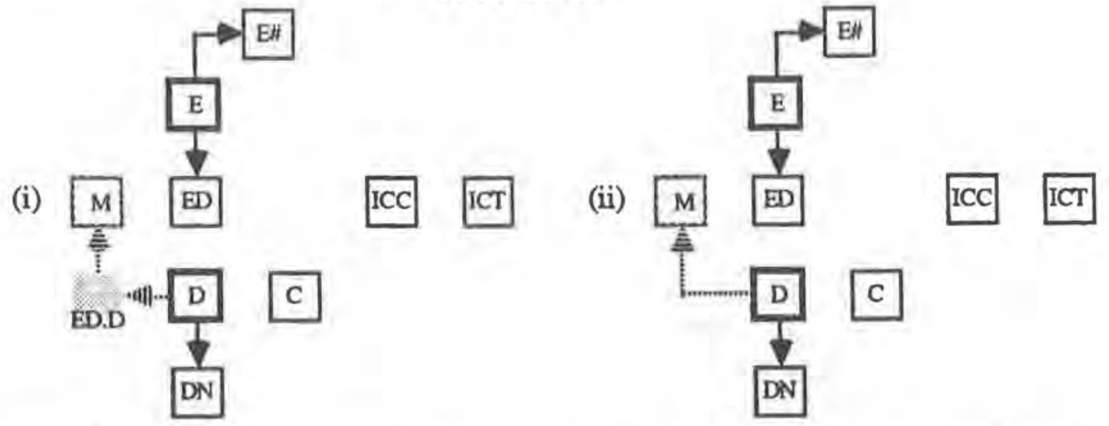
It is only in the calculation of degrees for defined sets that any significant human intervention is needed in the domain graph method of table design. The development of an algorithm for calculating the degrees of most of the defined sets declared in a typical set schema remains a research challenge.

3.2. <1,1>- and 1-subgraphs of an Augmented Domain Graph

It is now assumed that each edge of an augmented domain graph is labelled with a pair of degrees. A **<1,1>-subgraph** of an augmented domain graph is a connected subgraph with edges just those that are labelled with the degrees <1,1>. A **1-subgraph** is a connected subgraph with edges just those that are labelled with the lower degree 1. A <1,1>- or 1-subgraph is **maximal** if it cannot be enlarged by the addition of nodes that are connected by edges labelled with <1,1>, respectively the lower degree 1. It is elementary that the maximal 1-subgraphs of a domain graph provide a unique partition of its nodes, as does also the maximal <1,1>-subgraphs. Further, the nodes of each maximal 1-subgraph are partitioned by the maximal <1,1>-subgraphs. Only maximal 1- and <1,1>-subgraphs need be used in table design, although nonmaximal ones may be used.

In figure 3.4 (i) some of the <1,1>-subgraphs of the augmented domain graph for the set schema for Simple University are illustrated.

FIGURE 3.4



Examples of 1-subgraphs are obtained if the missing edge from D to ED, which is labelled with the degrees <1,*>, is replaced, or the node labelled with EDC.C is restored with its edges from C and to ICC.

The choice of whether tables should be constructed from the 1-subgraphs or the <1,1>-subgraphs is the second human intervention needed in the domain graph method of table design. Motivations for this choice will be discussed in section 3.7. In the meantime, the method will be illustrated using <1,1>-subgraphs.

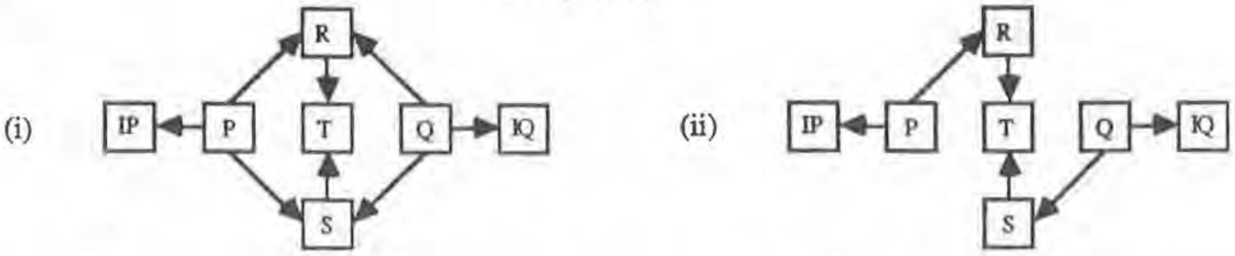
Not all <1,1>-subgraphs of the augmented domain graph are illustrated in figure 3.4 (i). The only subgraphs illustrated are those that contain a node labelled with a set whose membership is to be recorded. The membership of value sets need not be recorded, nor is it necessary to record the membership of defined sets such as EDC, ENA, and IAD, since they were needed only as domains for base sets, so the subgraphs containing nodes labelled with these sets have been dropped. Also the nodes labelled MA and MNA have been dropped because the membership of these sets can be obtained from M. They could, of course, be retained if it is desirable to have all base sets represented as tables.

In figure 3.4 (ii) one of the subgraphs of (i) has been simplified by eliminating the node labelled with the undeclared set ED.D, and replacing the directed path via the node with a single edge, as called for in step (2) of the method. The result is a graph that is no longer a subgraph of the augmented domain graph. Since each edge of the directed path from the node labelled D to the node labelled M has lower degrees <1,1>, the new edge introduced has those degrees also. It is these simplified subgraphs that will be used to determine tables.

3.3. Breaking Undirected Cycles in Subgraphs

All of the subgraphs of figure 3.4 (ii) are undirected acyclic; that is, they are trees. If one was not a tree, then it would be necessary to remove edges to make it so, as illustrated in the next figure.

FIGURE 3.5



Although the cycles of (i) could be broken by dropping any two edges that leaves the graph connected, the edges chosen to form (ii) have tails that are the bottom nodes labelled P and Q; that is, they are nodes of the cycle that are not the head of an edge of the cycle. To select other edges to break cycles results in a table with more columns than are necessary. However, any pair of edges that break the cycles, and that have tails the bottom nodes labelled P and Q, can be selected.

The trees obtained from step 2, and where necessary step 3, will be called henceforth the **selected trees**.

3.4. The Identifier Extension of a Selected Tree

A member of a value set can always be recorded in a table since it can be read by both humans and machines. A member of other sets can only be indirectly recorded in a table by recording an identifier for it. For example, a table of members of E consists of all the employee numbers that had been assigned to members of E. Similarly, a table of the names of departments in D is used to record the members of D. A table of the members of a nonprimitive set consists of the tuples of identifiers of the tuples that are members of the set; for example, a table for ED consists of the pairs <e#, dn> for which the employee with employee number e# has been ED associated with the department with name dn.

To construct a table from a selected tree it is necessary to extend the tree in order to ensure that every set that labels a node of the tree is provided with its identifier. The resulting tree is called the **identifier extension** of the selected tree. An algorithm for constructing the identifier extension for any selected tree will be described.

Recall from 2.10 that an arity predecessor of a nonprimitive set is any immediate domain predecessor of the arity domain of the set, and that the multiplicity of an arity predecessor is the multiplicity of it for the arity domain. When edges are dropped from the augmented domain graph to form a forest of trees, a nonprimitive set labelling a node of the graph can become disconnected from one or more of its arity predecessors. For example, in figure 3.4 (ii), each of the sets E#, ED, DN, and C is its own arity domain and was disconnected from one or more of its arity predecessors, which are in these cases immediate domain predecessors: E# was disconnected from VE#, ED from D, DN from VN, and C from both D and VC#. In addition each of the sets M, ICC, and ICT, with arity domains ED, EDC, and EDC, respectively, was disconnected from one or more of its arity predecessors: M was disconnected from E, and both ICC and ICT were disconnected from both E and C.

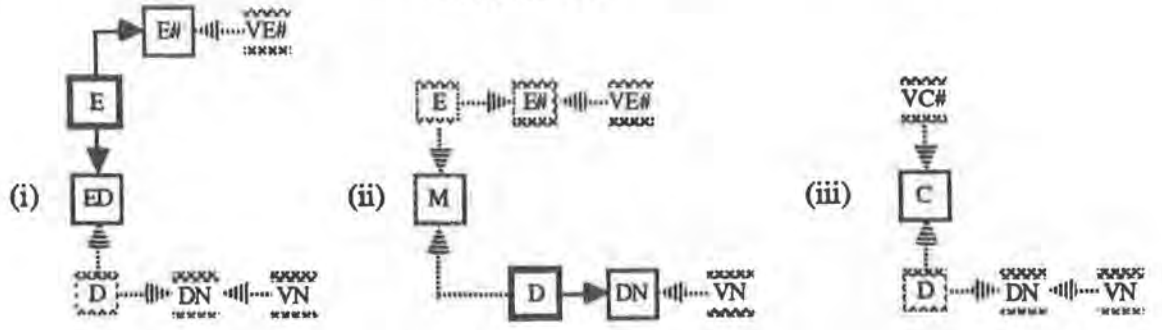
In order to ensure that every member of every set labelling a node of a selected tree can be given an identifier, it is necessary to first extend the tree to one in which each node nde is **arity predecessor complete**; that is to say, nd satisfies the following condition: Let S be a set of arity m labelling nd, and let $S_1, \ldots, S_m$ be the arity predecessors of S, with repititions appropriate for the multiplicities. Then there are nodes $nd_1, \ldots, nd_m$, labelled respectively with $S_1, \ldots, S_m$, and such that for each nd_i, $<nd_i, nd>$ is an edge of the tree.

By beginning with a selected tree, and repeatedly adding as needed new nodes nd' and edges <nd', nd>, with nd' labelled with an arity predecessor of the set labelling nd, an extension of the tree can be obtained that is arity predecessor complete for each of its nodes.

Each bottom node nd of the resulting tree will be labelled with a primitive base set, or a value set. Nothing more need be done for a value set since it is its own identifier. For a primitive base set S, an identifier must be provided. Let S be a primitive base set labelling nd, and let IS be its identifier with value set VIS. Nodes nd' and nd" labelled with IS and VIS respectively are added to the tree together with the edges <nd, nd'> and <nd", nd'>, if such nodes do not already exist.

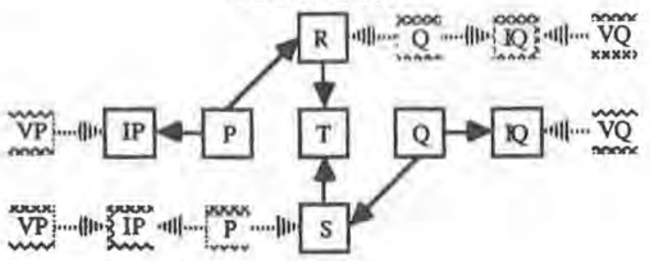
The identifier extensions of three of the selected trees illustrated in figure 3.4 (ii) are illustrated in figure 3.6. The nodes that have been added to the selected trees to form their identifier extensions are all represented by cross-hatched boxes.

FIGURE 3.6



The identifier extension of the tree (ii) of figure 3.5 is illustrated in figure 3.7. It is assumed that IP is an identifier for P with value set VP, and that IQ is an identifier for Q with value set VQ.

FIGURE 3.7



3.5. The Table for a Selected Tree

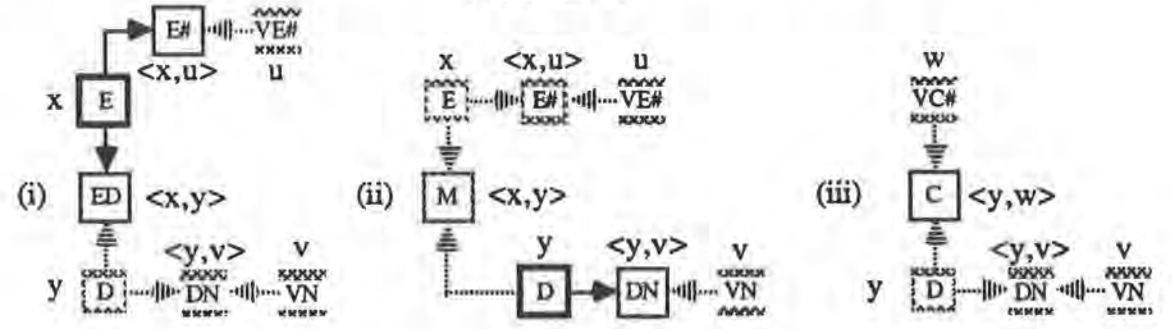
The declaration of a defined set, and therefore of a table for a selected tree, has the form of the declaration of EDC in section 2.9. The two formal parts needed for the declaration are the intension of the set expressed as an assertion of DEFINE, and the domain assertions that simultaneously declare the domain and the ranges of the variables appearing in the assertions.

The intension of the defined set that is the table for a given selected tree is obtained in two steps. A conjunction of elementary assertions, called a **join assertion** of the tree, is determined from the identifier extension of the tree. To form a join assertion, distinct variables must be assigned to each of the bottom nodes of the identifier extension, and then these variables, or nested tuples of them, are assigned to the other nodes in a manner described below. Each elementary assertion of the join assertion is then of the form tup:S, where S is a nonprimitive set labelling a node of the identifier extension that is not a bottom node, and tup has been assigned to the node. The intension of the table for the given selected tree is formed from a join assertion by prefixing it with an existential quantifier for each variable assigned to a node that is labelled with a primitive base set.

The domain assertions all take the form var:VL, where VL is a value set labelling a bottom node of the tree, and var is the variable assigned to the bottom node. The order of the domain assertions determines the order of the columns appearing in the table and is unrestricted. A convenient order is to have the identifier(s) for a set labelling a node preceed attributes of the set and associations with other sets. Since each such set labels a node of exactly one selected tree, the order can be determined automatically.

The process of assigning variables and nested tuples of variables to the nodes of the identifier extension of a selected tree is a simple one that can best be described by examples. In figure 3.8, variables and tuples have been assigned to the nodes of the trees of figure 3.6.

FIGURE 3.8



Consider, for example, the tree (ii). The nodes labelled E, VE#, D, and VN, are the bottom nodes of the tree. They have been assigned the distinct variables x, u, y, and v. The node labelled E# is assigned <x,u> because the node labelled E has been assigned x and the node labelled VE# has been assigned u, and the domain of E# is ExVE#. Similarly for the node labelled DN. The node labelled M is assigned <x,y> because the node labelled E is assigned x, the node labelled D is assigned y, and the domain of the arity domain ED of M is ExD.

Variable assignments to the tree illustrated in figure 3.7 can be given in a similar fashion. For example, let the variables ur and us be assigned to the two nodes labelled VP, the variables vr and vs to the two nodes labelled VQ, the variables xr and xs to the two nodes labelled P, and the variables yr and ys to the two nodes labelled Q, all in order from top to bottom in the diagram. Assume that the domain of R and of S is PxQ, and that the domain of T is RxS. Then the tuple assigned to the node labelled T is << xr,yr>,< xs,ys>>.

The tables corresponding to the trees in figure 3.8 are:
TE={ u:VE#, v:VN | **[For some** x:E, y:D] (<x,u>:E# **and** <x,y>:ED **and** <y,v>:DN) | },
TD={ v:VN, u:VE# | **[For some** x:E, y:D] (<x,u>:E# **and** <x,y>:M **and** <y,v>:DN) | }, and
TC= { v:VN, w:VC# | [**For some** y:D] (<w,y>:C **and** <y,v>:D) | }.
In a more realistic example the tables would have additional columns for the values of attributes of declared sets. However, the table TE would not have columns for the attributes of D, since such attributes would never label nodes of the selected subgraph containing the original node labelled with E. Similarly the attributes of D would appear only in TD, and the attributes of C in TC. The tables obtained from the other two selected trees of figure 3.4 are declared:
TICC={ u:VE#, v:VN, w:VC# | **[For some** x:E, y:D] (<x,u>:E# **and** <y,v>:DN **and** <x,<y,w>>:ICC) | }, and
TICT={ u:VE#, v:VN, w:VC# | **[For some** x:E, y:D] (<x,u>:E# **and** <y,v>:DN **and** <x,<y,w>>:ICT) | }.
The table obtained for the tree of figure 3.7 under the variable assignment given earlier is
TPQRST={ ur:VP, vr:VQ, us:VP, vs:VQ | **[For some** xr:P, yr:Q, xs:P, ys:Q] (<xr,ur>:IP **and** <xs,us>:IP **and** <yr,vr>:IQ **and** <ys,vs>:IQ **and** <xr,yr>:R **and** <xs,ys>:S **and** <<xr,yr>,<xs,ys>>:T) | }.

3.6. Justification for the Method

Consider any set schema Sch. Let S be a set declared in Sch of interest to users. S labels exactly one node of the domain graph of Sch, and therefore exactly one node nde of the augmented domain graph. Let Tdg be the single tree obtained in step 3 of 3.1 of which nde is a node. Let Tr be the identifier extension of Tdg obtained in step 4, and let T(Tr) be the table obtained in step 5. If the domain graph method is correct, then it should be possible for a user to determine the membership of S from the table T(Tr).

Consider, for example, the table TE. E and ED are the only two sets of interest to users of TE that label nodes of the selected tree of figure 3.4 (ii) for which the tree (i) of 3.6 is the identifier extension. From knowledge of E# and DN, a user can determine from TE the membership of two related sets EU and EDU declared as follows:

EU={ x:E | [**For some** u:VE#,v:VN] (<u,v>:TE **and** <x,u>:E#) | }, and
EDU={ <x,y>:ExD | [**For some** u:VE#,v:VN] (<u,v>:TE **and** <x,u>:E# **and** <y,v>:DN) | }.
As far as the user is concerned, the table TE is correct for E if EU has the same extension as E, and EDU has the same extension as ED. Similarly TC is correct for C if the set
CU={ <y,w>:DxVC# | [**For some** v:VN] (<v,w>:TC **and** <y,v>:DN) | }
has the same extension as C.

In [Gil87] a definition is given of the user form SU of S, of which EU, EDU, and CU, are special cases, and the following theorem is proved:

> **Theorem:** Assume that all declared and defined degree constraints labelling edges of an augmented domain graph are satisfied by the membership of the declared sets. Let S be a declared set labelling a node of a 1-connected, not necessarily maximal, subtree of the domain graph, and let SU be the user form of S declared for the table obtained from the subtree. Then S and SU have the same extension.

The statement and proof of a simple sufficient condition on the satisfiability of the degree constaints, is also given there.

3.7. 1-Connected or <1, 1>-Connected Selected Trees?

Since tables obtained from 1-connected or <1,1>-connected subgraphs are equally correct, the decision as to which tables to use depends upon other considerations, sometimes upon taste.

When the tables are declared as defined actual, and correspond to flat file data structures that record them, duplication of data is a concern. Using 1-connected subgraphs can result in tables with unnecessary duplication of data, and it is therefore prudent to consider <1,1>-connected subgraphs. Because the <1,1>-connected subgraphs partition the nodes of the 1-connected subgraphs, the tables obtained from the <1,1>-subgraphs slit the tables otained from the 1-connected subgraphs. Therefore using <1,1>-connected subgraphs results in the construction of more than the minimum number of tables, and therefore in the unnecessary duplication of data. A simple calculation will determine which kind of duplication is the least demanding of space.

When the tables are used purely as presentation data structures, the choice is more fully a matter of taste. Taste need not be restricted to the unnecessary, and often artificial first normal form. Tables that are not in that form can be easily defined, and the additional formatting provided for data can make them more comprehensible to users. The restriction to first normal form demanded in the relational model results, after all, from a burdening of a presentation view of data with implementation concerns. Other evidence of that burdening is the necessity to declare keys for tables in addition to the identifiers declared for the primitive base sets upon which the tables are based.

It is interesting to note that the synthetic method of table design described in [Bern76], which also found a minimum number of tables, is dependent upon functional dependencies for attributes that can be defined in terms of the pairs <1,1> of degrees. The domain graph method is also related to the method described in [Knt83a]. Its relationship to the method described in [TYF86] is less clear because of the use of normalization in that method.

4. Integrity Constraints

Only two kinds of integrity constraints are expressible in the SET model, the constraints implicit when one set is declared to be the domain of another, and the degree constraints declared for base sets. As explained in 2.11, the two interact. For example, the effect of the degree constraints of ICT are described in terms of its arity domain EDC and its arity predecessors E and C. In 4.1 the effect of these constraints on the tables declared in 3.5 is described, while in 4.2 the possibility of using the fanset data structure of the network model to maintain the constraints is discussed.

4.1. The Relational Model

The domain graph method of designing tables from a set schema for an enterprise results in a relational schema for the enterprise. Since the tables are defined sets, the membership of them is automatically maintained by a management system supporting the set schema. From the definitions of the tables can be determined constraints that will be automatically maintained by the management system, but that will have to be declared for a relational database.

Consider, for example, the constraints that are determined by the definitions of the tables TD, TE, TC, TICC, and TICT given in 3.5:

1. the VE# values of TD, TICC, and TICT, must appear in TE;
2. the VN values of TE, TC, TICC, and TICT, must appear in TD;
3. the VC# values of TICC and TICT must appear in TC;
4. the <VN, VE#> values of TD must appear reversed in TE;
5. the <VE#, VN> values of TICC must appear in TE;
6. the <VN, VC#> values of TICC must appear in TC;
7. for those VN values appearing in TICC, the <VN, VE#> values of TD must appear in TICC; and
8. the <VE#, VN, VC#> values of TICT must appear in TICC.

The constraints 1-4 can be seen to be determined from the identifier extensions for the trees from which the definitions of the tables were obtained. The identifier extensions from which TE, TD, and TC were obtained are illustrated in figure 3.8. Note that the node labelled D in (i) was added while forming the identifier extension (i) of a selected subtree of the domain graph. Similarly for the nodes labelled E in (ii) and D in (iii). There is only one set E and one set D in the set schema for SU. Consequently any value of VE# appearing in TD must be among the values of VE# appearing in the table TE. Similarly any value of VN appearing in TE or TC must be among the values of VN appearing in the table TD. But also the <VN, VE#> values of TD must appear reversed in TE, since every pair that is a member of ED.D must be a member of ED. The constraints involving TICC and TICT are obtained in a similar fashion.

The given constraints are all examples of what has been called referential integrity constraints [Date83], although they are much more complicated than those discussed in the literature. Nevertheless, were the tables TE, TD, TC, TICC, and TICT, to be declared for a relational database, the management system for the database would have to maintain these constraints.

The fact that the constraints do not have to be explicitly recognized in the set schema for SU, but that they follow implicitly from the declarations of the tables as defined sets, indicates one advantage a conceptually oriented model such as SET has over a presentation oriented model such as the relational. Although the domain and degree constraints of the SET model express simple real world constraints in a simple fashion, the form that these constraints take in the defined tables is much less transparent. In more complex databases, such as those supporting a kernel schema or knowledge bases, the number of such inclusion constraints that have to be maintained overwhelms any benefits that can be expected from the presentation oriented relational database model.

4.2. Fansets and Referential Integrity

In [Date83] the use of the fanset data structure in the maintenance of referential integrity constraints is discussed. With an appropriate pointer implementation, the constraints 1-4 can be maintained using fansets, although the reversal in the fourth adds a complication. The constraints 5 and 6 present much greater difficultes, while 7 is of a kind not customarily implemented using fansets. The difficulties presented by 5 and 6 arise from the fact that the two pairs <VE#, VN> and <VN, VC#> have VN in common; although a single pointer can maintain either one of these constraints, a pair of pointers will not maintain the pair. It is therefore not clear how the tables would be implemented in the Network Model. On the other hand, a fanset implementation of a set schema may be feasible if defined sets can be maintained virtually.

5. Conclusions

The presentation orientation of the relational model provided a more abstract view of data than could be provided by the hierarchical or network models. The model did not, however, completely free a user from implementation concerns. Its emphasis on first normal form and the need for keys are consequences of its treatment of tables as flat file storage structures, regardless of whether they are actually to be so used. More importantly, the concentration of the relational model on data, rather than on the reality the data is intended to describe, has resulted in unnecessarily complicated table design methods and integrity constraints. Through the use of the specification/query language DEFINE of the SET model, the design of presentation data structures such as tables can be almost fully automated, declared as defined sets, and proved to be correct. As a consequence integrity constraints for presentation data structures need not be stated, but are maintained by any system that maintains the domain and degree constraints of the SET model.

BIBLIOGRAPHY

[Bach69] BACHMAN, C.W. Data structure diagrams. *Data Base* 1,2 (Summer 1969), 4-10.

[Bern76] BERNSTEIN, P.A. Synthesizing third normal form relations from functional dependencies. *ACM Trans. Database Syst.*, 1, 4 (March 1976), 271-298.

[Blac85] BLACK, MICHAEL JULIAN. Naive Semantic Networks. Final Paper, Directed STudy in Computer Science. Dept of Comp. Sci., Univ. of B.C. Jan 22, 1985.

[BMS84] BRODIE, MICHAEL L., MYLOPOULOS, JOHN, AND SCHMIDT, JOACHIM W. On conceptual modelling, Springer-Verlag, 1984.

[Chen76] CHEN, PETER PIN-SHAN. The Entity-Relationship model - toward a unified view of data. *ACM Trans. Data Base Syst.*, 1, 1 (March 1976), 9-36.

[Chen77] CHEN, PETER PIN-SHAN. The Entity-Relationship model - A basis for the enterprise view of data. AFIPS Conference Proceedings, Vol. 46, 1977 NCC.

[Chen85] CHEN, PETER P.S. Database Design Based on Entity and Relationship. [Yao85], 174-210.

[Codd79] CODD, E.F. Extending the Database Relational Model to Capture More Meaning. *ACM Trans. Database Syst.*, 4, 4 (Dec 1979)

[Date83] DATE, C.J. An Introduction to Database Systems, Vol.II. Addison-Wesley, 1983.

[DJNY83] DAVIS, CARL G., JAJODIA, SUSHIL, NG, PETER ANN-BENG, AND YEH, RAYMOND T. Entity-relationship approach to software engineering, North-Holland, 1983.

[DKM86] DE TROYER, O., KEUSTERMANS, J., AND MEERSMAN, R. How Helpful is an Object-Oriented Database Model?. [DiDa86]. 124-132.

[Ditt86] DITTRICH, KLAUS R. Object-oriented Database Systems: The Notion and the Issues. (extended abstract) 1986 International Workshop on Object-Oriented Database Systems. 2-4.

[FuNe86] FURTADO, ANTONIO L. AND NEUHOLD, ERICH J. Formal Techniques for Data Base Design. Springer-Verlag. 1986.

[Gil77] GILMORE, PAUL C. Defining and computing many-valued functions. Parallel Computers - Parallel Mathematics. FEILMEIER, M. (ed.), North-Holland (1977), 18-23.

[Gil86a] GILMORE, PAUL C. Natural deduction based set theories: a new resolution of the old paradoxes. *J. Symb. Logic*, 51, 2 (June 1986), 393-411.

[Gil86b] GILMORE, PAUL C. Class notes for CPSC 404. Dept of Computer Science, Un. of B.C. August 11, 1986.

[Gil87] GILMORE, PAUL C. Justifications and Applications of the SET Conceptual Model.Dept of Computer Science Tech. Report 87-9, Un. of B.C. April 1987.

[HaMc81] HAMMER, MICHAEL, AND McLEOD, DENNIS. Database description with SDM: a semantic database model. *ACM Trans. Database Syst.*, 6, 3 (Sept 1981), 351-386.

[Knt78] KENT, WILLIAM. Data and Reality, North-Holland, 1978.

[Knt81] KENT, WILLIAM. Consequences of Assuming a Universal Relation. *ACM Trans. Database Syst.*, 6, 4 (Dec 1981), 539-556.

[Knt83a] KENT, WILLIAM. Fact-based data analysis and design, [DJNY83], 3-54.

[Knt83b] KENT, WILLIAM. The Universal Relation Revisited. *ACM Trans. Database Syst.*, 8, 4 (Dec 1983), 644-648.

[KhCo86] KHOSHAFIAN, SETRAG N. AND COPELAND, GEORGE P. Object Identity. [Meyr86]. 406-416.

[LeSa83] LENZERINI, MAURIZIO, AND SANTUCCI, G. Semantic integrity and specifications, [DJNY83], 529-550.

[LuKl86] LUK, W.S. AND KLOSTER, STEVE. ELFS: English Language for SQL. *ACM Trans. Database Syst.*, 11, 4 (Dec 1986), 447-472.

[LyKe86] LYNGBACK, PETER, AND KENT, WILLIAM. A Data Modelling Methodology for the Design and Implementation of Information Systems. [DiDa86]. 6-17.

[Morr] MORRISON, RODERICK. Implementating a Set Based Data Model and its Data Definition/Manipulation Language. PhD thesis, Department of Computer Science, Un. British Columbia. *In progress*.

[MyWo80] MYLOUPOLOS, J., AND WONG, H. Some features of the TAXIS data model. *Proc. 6th Int. Conf. Very Large Databases*, Montreal (1980).

[Rock81] ROCK-EVANS, ROSEMARY. Data analysis. IPC Electrical-Electronic Press Ltd, Sutton, Surrey, England (1981).

[SAAF73] SENKO, M.E., ALTMAN, E.B., ASTRAHAN, M.M., AND FENDER, P.L. Data structures and accessing in data-base systems. *IBM Syst. J.* 12, 1 (1973), 30-93.

[Ship81] SHIPMAN, DAVID W. The functional data model and the data language DAPLEX. *ACM Trans. Database Syst.*, 6, 1 (March 1981), 140-173.

[Sowa84] SOWA, J.F. Conceptual Structures: Information Processing in Mind and Machine. Addison-Wesley, 1984.

[TYF86] TEOREY, TOBY J., YANG, DONGQING, AND FRY, JAMES P. A Logical Design Methodology for Relational Databases Using the Extended Entity-Relationship Model. ACM Computing Surveys, 18, 2 (June 1986). 197-222.

[TaFr76] TAYLOR, ROBERT W.; AND FRANK, RANDALL L. CODASYL Data-Base Management Systems. *ACM Comp. Surveys.* 8,1 (March 1976), 67-103.

[TrLo87] TRYON, D.C.; AND LOYD, D.G. Information Resource Depository: History, Current Issues, and Future Directions. Pacific Bell, A Pacific Telesis Company. Presentation to Canadian Information Processing Society, Vancouver, Canada, February 1987.