The Semantic Database Model
As A Basis For An
Automated Database Design Tool

Sonia Berman

A Thesis prepared under the Supervision of
Professor K.J. McGregor
submitted in fulfilment of the requirements for
the degree of Master of Science in Computer Science
at the University of Cape Town

May 1983

ABSTRACT

The automatic database design system is a design aid for network database creation. It obtains a requirements specification from a user and generates a prototype database. This database is compatible with the Data Definition Language of DMS 1100, the database system on the Univac 1108 at the University of Cape Town. The user interface has been constructed in such a way that a computer-naive user can submit a description of his organisation to the system. Thus it constitutes a powerful database design tool, which should greatly alleviate the designer's tasks of communicating with users, and of creating an initial database definition. The requirements are formulated using the semantic database model, and semantic information in this model is incorported into the database as integrity constraints. A relation scheme is also generated from the specification. As a result of this research, insight has been gained into the advantages and shortcomings of the semantic database model, and some principles for 'good' data models and database design methodologies have emerged.

ACKNOWLEDGEMENTS

I would like to express my thanks to my supervisor, Professor K.J.McGregor, for his advice and support. The encouragement he gave me throughout this project was very greatly appreciated.

Thanks are also due to Mr. D. Roberts and Mr. P. Loebenberg for the use of their Hewlett Packard 9845B on which the diagrams of this thesis were prepared.

I would like to thank my husband for draughting the diagrams, and for his patience.

Table of contents

Table of contents

Table of contents

Table of contents

1. CHAPTER 1. INTRODUCTION.

Database design is a formidable task which continues to present problems to theoreticians and practitioners alike. No universally-accepted database design methodology exists, and currently the major part of database research is directed towards solving this problem. Some helpful results are beginning to emerge from this work, for example normal form theory. This is based on a sound mathematical foundation and provides guidelines for relational database design. In the realm of network databases, three types of design aid exist. These are data models, prototype databases and automated design tools.

Data models are a means of representing data and their meaning in a natural and easy, yet precise, manner. Whereas schemas are computer-oriented data descriptions, data models are people-oriented. A data model of an enterprise simplifies the process of requirements specification and facilitates the choosing of a database structure. Prototypes are initial 'test' databases used to detect misconceptions and omissions in the specification. Experimentation with a prototype can highlight potential problems and indicate where envisaged transactions cannot be supported, or performance requirements cannot be met. Some automated tools exist to assist a designer with parts of the design process, such as requirements gathering, database structure creation, database performance evaluation or physical parameter allocation [Hammer 1977,Tompa 1976,Jankowitz 1982,Gerritsen 1975,Hubbard 1979 and Roussopoulos 1979]. Generally these only serve as a guide to the designer, who must analyse and modify their results.

The "automated database designer" (henceforth called ADD) was

undertaken to integrate in one system all these three facilities. Hence ADD automates the database design process from information collection and model specification through to the construction of a prototype network database. The prototype incorporates comprehensive integrity checks, in an attempt to support as much as possible of the semantic information in the model. The creation of a relation scheme was also implemented. The system generates several listings indicating the design decisions taken. These can be examined by the human designer to determine the effect of specific requirements. They should also facilitate the writing of the final working schema.

The model chosen for ADD was the Semantic Database Model [Hammer 1978, Hammer 1981, King 1982]. This model was considered the best model for the undertaking, because it has several powerful modelling features and yet is simple and easy to use. A second objective in implementing ADD was to investigate the effectiveness of SDM for database design.

Although the system is described in some detail in the succeeding chapters, a summary of the process of database synthesis is now presented. This is intended to acquaint the reader with the system; unfortunately it requires using terminology which will be explained in the next chapter: this is unavoidable.

To create the network, a record type is formed for each entity type, comprising its non-repeating attributes. Repeating attributes constitute a separate record type, connected to the entity they characterise as member of a set. A functional dependency is also separated out to form a new record type, and is made the owner of a set linking it to the entity type it described. Relationships translate into sets, with their

one:one/one:many/many:many nature distinguishing the owner from the member. A confluent hierarchy is used for the many:many case.

Generalisation or IS-A hierarchies are also treated. This refers to entities which are special types, or subtypes, of other entity types (such as OILTANKERS, special SHIPS). If the subtype has fields not present in its "supertype", or if it participates in its own individual relationships, it is represented by a separate record type. This is linked to the "supertype" as the member of a set, and the construction of these sets is controlled by specifying appropriate CHECK and RESULT clauses. The semantic information in the model, such as the description of derived data, is used to refine the network structure and also causes integrity constraints to be incorporated. A simple set of physical parameters completes the database definition; the only complexity that can occur here is with specifications of multi-level paths in SET SELECTION clauses. The finer points of the ADD design methodology are deferred until chapter seven, as they require a thorough understanding of SDM features.

In order to justify the development of the ADD project, the next section outlines the difficulties of database design and then relates this to the objectives of the system. Finally, an overview of ADD is given to indicate how it sets about achieving these goals.

1.1. Motivation.

Database design is a difficult task, for several reasons. The most important are users' inability to provide a good specification; the difficulty in obtaining an adequate logical structure; and the

problem of choosing physical properties to gain optimal (or at least adequate) performance. (The logical structure describes only the data and their associations, the physical parameters augment this with storage and access strategies, file descriptions, etc.)

These problems arise because of inadequacy on the part of both users and database designers. Frequently the user himself doesn't know exactly how his database will be used nor does he fully understand the data items and their inter-relationships. The problem is aggravated if he cannot distinguish important information from irrelevancies. Requirements specifications are thus frequently incomplete, inconsistent, or contain misconceptions. In addition, there are often communication problems between the database analyst and user, and a specification may be misinterpreted. A data model forces users to think clearly and carefully and thus they understand and describe their data better. "People use data models to abstract and understand concepts about data and information" [Tsichritzis 1982].

Even if the information gathering is correct, complete and well-understood, the designer can still have difficulty in providing an efficient system. There is a lack of database expertise, particularly in database design. Computer professionals are accustomed to thinking in terms of conventional files and of designs based on processing applications; they have difficulty adapting their modes of operation to the database environment. If the design is based on processing requirements, not on the fundamental structure and properties of the data itself, problems arise when transactions change and the structure

can no longer support them adequately. Design difficulties arise because there are so many options available when deciding both logical and physical characteristics. If the designer is too preoccupied with efficiency, the logical structure may not meet with user requirements. A poor database design generally proves to be an extremely costly mistake. The main reason for this is that operational databases are difficult to reorganise, especially if very large.

The ADD system was intended to assist the designer in the following ways. It should virtually eliminate the problem of obtaining a good, unambiguous specification. It should greatly reduce the difficulties which face the designer of the logical structure, by providing him with one that will require minimal, if any, alterations. It should also help to some extent with physical property assignment. Before describing how this was carried out, some background information on database design and data models is presented. This is explained in the next section, which gives a preview of the content and structure of this thesis.

1.2. Thesis Overview.

It is assumed that the reader is familiar with the concept of a database and the workings of a database system. As a foundation to what follows however, chapter two presents a brief outline of databases and relational database theory, before reviewing current database design methodologies and techniques. An extensive literature survey showed that no universally-accepted database design methodology exists at present and that designers accordingly have to rely largely on their own intuition.

Amongst the design aids mentioned, the use of data models was one of the most significant. A survey of data models was undertaken, in order to establish the most suitable one for the ADD system. Chapter three describes and evaluates these models; while chapter four is a detailed presentation of SDM, which also explains why this particular model was chosen.

Having established that an SDM model would be the initial objective and a prototype schema the final goal, two distinct phases of the system were distinguished: model construction and schema generation. (A schema is a database definition that describes the structure and organisation of a database.) Chapter five describes how the first phase was implemented. It was decided that the model should be designed by the users, as only they are intimately familiar with the enterprise, its workings and its data. As these are non-computer people, a suitable user interface had to be created for this phase. Several articles on "user friendliness" in man-computer dialogues were studied, to discover the techniques to employ. Some of the interesting and useful ideas discovered, are also described in chapter five.

When implementing phase two, it was decided to incorporate the conversion of the model to a relation scheme as well as to a network database. The reasons for this are firstly that the design of a relation scheme is a useful step in a database design process. Secondly, relational databases are emerging as the most significant type of database, with most of the research being directed towards such systems. To investigate the potential of SDM as a basis for database design, it was considered essential to analyse its adaptibility to relation scheme synthesis, and not only to networks. Chapter six explains how an SDM model is

translated into a collection of relations and briefly describes the program that implements this. The resulting relation scheme is analysed and evaluated within the framework of normal form theory.

This relation scheme together with the SDM model of the enterprise is used to design the network database. Each relation is converted into a record or set, and then the semantics of the data model are used to refine this structure and introduce integrity constraints. Finally, physical parameters are included and the complete prototype database definition is generated. The creation of a network database and the incorporation of SDM features into this, is described in chapter seven. The components of the ADD system can be seen in fig. 1.1.

Chapter eight discusses how the synthesised database can be tuned to gain better performance. The emphasis here is on adjustment of physical parameters, as the prototype should have a satisfactory logical structure. For completeness however, suggestions for alterations to the latter are also given.

Having implemented both phases of the ADD system, considerable insight has been gained into the appropriateness of SDM as a basis for database design. This is described in chapter nine, where its desirable features as well as its shortcomings are discussed, and suggestions for a modified SDM are presented. Chapter ten concludes this thesis with an appraisal of the ADD system and the establishment of some criteria for 'good' data models and database design methodologies.

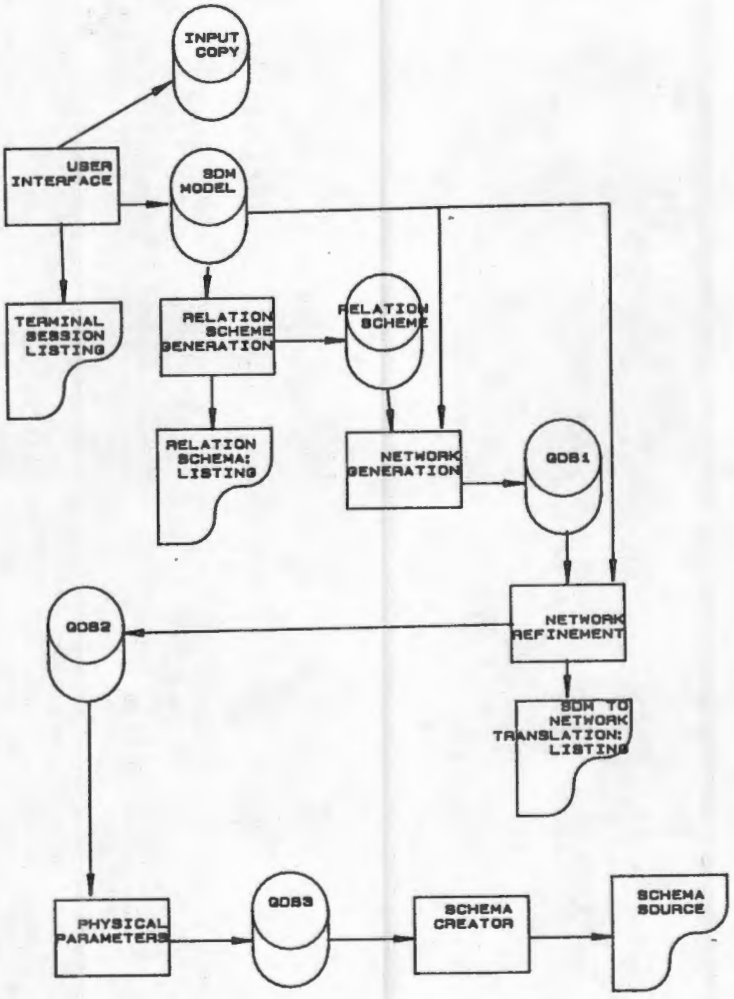


Figure 1.1 The ADD System Modules.
The database being designed is called the QDB. Here QDB2 and QDB3 represent successive modifications to the initial structure, QDB1.

[Gerritsen 1975] Gerritsen, R. "A Preliminary System for the design of DBTG Data Structures", Comm. of the ACM, Vol. 18, 1975, pp. 551-557.

[Hammer 1977] Hammer, M. "Self-Adaptive Automatic Database Design", AFIPS 77, 1977, pg. 123.

[Hammer 1978] Hammer, M. and McLeod, D. "The Semantic Database Model: A Modelling Mechanism for Database Applications", Proc. of the ACM SIGMOD Int. Conf., Austin TX, May 1978.

[Hammer 1981] Hammer, M. and McLeod, D. "Database Description with SDM: A Semantic Database Model", ACM Trans. on Database Systems Vol. 6, No. 3, pp. 351-386. [Hubbard 1979] Hubbard, G.U. "Computer-Assisted Logical Database Design", Computer-Aided Design, Vol. 11, No. 3, May 1979, pp. 169-179.

[Jankowitz 1982] Jankowitz, H., Kilfoil, P.W. and Rabkin, I. "DBTRACE: A High-Level Database Performance Monitoring and Debugging System", Technical Report, University of Cape Town, 1982.

[King 1982] King, R. and McLeod, D. "Semantic Database Models", in Database Design, (ed. Yao, S.B.), Prentice Hall, 1982.

[Roussopoulos 1979] Roussopoulos, N. "Tools for Designing Conceptual Schemata of Databases", Computer-Aided Design, Vol. 11, No.3, May 1979, pp. 119 - 120.

[Tompa 1976] Tompa, F.W. "Choosing an Efficient Internal Schema", in Systems for Large Databases, (eds. Lockemann and Neuhold), North-Holland, 1976.

2. CHAPTER 2. DATABASES AND DATABASE DESIGN.

As it is assumed that the reader has some knowledge of databases, This chapter very briefly describes what databases are and why they are used. The three types of database system are reviewed, with emphasis on the network model, and then the theory of relational database design is discussed. The final section presents a survey of current database design methodologies and techniques.

2.1. Introduction to databases.

A database can be defined as a "collection of interrelated data stored together without harmful or unnecesary redundancy to serve one or more applications in an optimal fashion" [Martin 1975]. It is essentially a large collection of data on secondary storage, which is controlled by special software called the DBMS (database management software or database management system.) "Database technology is concerned with both the physical and the logical organisation of the data. The physical organisation refers to the way the data is recorded on a storage medium. The logical organisation, or the data model, is the user's view of what data is stored on the database, and the relationships between the individual items of data" [Jankowitz 1982]. The DBMS performs all data manipulation, hiding implementation details from programmers, who need not be aware of the physcial organisation of a database.

A great deal of research is being undertaken in the field of database technology, and databases are becoming increasingly evident in commerce and industry. Why are databases so popular?

There are several advantages that database systems have over conventional files. Some excellent discussions on this can be found in [Date 1981,Kroenke 1977, Deen 1977,Ullmann 1980,Bradley 1982]. The major advantages of databases are listed below.

(1) Data independence is achieved. This is the immunity of application programs to changes in storage structure and/or access strategy.

(2) There is a common and controlled approach to data access. All database manipulations are performed under the centralised control of the DBMS.

(3) Data can be shared by multiple users accessing the database concurrently.

(4) Data integrity, privacy and security can be maintained by the DBMS.

(5) Redundancy can be eliminated or reduced.

(6) The DBMS can log all activities, gathering usage statistics, keeping page copies for recovery purposes, etc.

The disadvantages of database systems are the space and time overheads associated with a DBMS, and the lack of database expertise amongst computer personnel.

2.2. ARCHITECTURE OF A DATABASE SYSTEM.

The task of a database designer is to write a schema for the enterprise. A schema is a global definition of the logical structure of a database. It lists the data items and their format, as well as the relationships between them. Most DBMSs also require some specification in the schema of the mapping to storage, such as file descriptions, designation of sort keys, and stipulation of storage/access methods. The schema is used only by

the database designers and the DBA (database administrator).

Subschemas define the logical view of the database as seen by a particular application program or group of application programs. Since an application program is in general only concerned with certain section(s) of the database, a subschema consists of a listing of those data items and data relationships which are needed by a program. Subschemata may overlap and each may be used by several application programs.

It is in terms of a particular subschema structure that commands are issued to the DBMS for database manipulation. Application programs consist of such commands embedded in a 'host' language, which is a conventional programming language. Most DBMSs also support a query language facility. A database query language is english-like and easy to use. In this way simple processing such as obtaining lists or counts of items can be accomplished without the need for an application program.

2.3. THE THREE MAJOR DATABASE SYSTEMS.

Conventional DBMS's can be categorised into three types, according to how they represent relationships between data. These are the hierarchical, network and relational approaches. Objects are essentially represented as records of zero, one or more fields (attributes) in all three systems.

In the hierarchical and network systems relationships are respresented as links between records. In the former, the resulting structure of inter-linked records forms a forest of trees - i.e. each record has one and only one parent (except for

the root of each tree, which has none). A network is an extension of a hierarchical system in which a record can be linked to more than one 'parent', resulting in a graph.

In contrast, a relational database can be conceptualised as a collection of tables ("relations"), with one row of the table for each record in the database. The columns of the relation represent the attributes. A relational schema consists of one or more relations. The formal definition of a relation is a collection of n-tuples of the form <a1,a2,...,an>, where each a1 is drawn from the same domain d1, each a2 is drawn from the same domain d2, etc. (These domains or value sets need not necessarily be distinct.) If two object types are related, their keys appear in the same table.

The advantage of the relational model is that it can be manipulated in a "non-procedural" manner, with data being accessed simply by stating its properties. In contrast, the other two systems are structurally intricate and highly "navigational", placing too much emphasis on access paths and jeopardising their data independence. They are "not suited for a conceptual representation since they combine information about the logical structure of data with access path description, leaving some ambiguity as to the exact nature of the former" [Zaniolo 1982a]. For this reason, such models "typically do not allow the easy definition of new transactions" [McLeod 1982]. Their rigid structure tends to impose a fixed interpretation on the data and their inter-relationships.

All conventional database systems suffer from a lack of semantic expressiveness and hence databases can often be misinterpreted. Their emphasis on record structures has been criticised by authors

such as Kent [Kent 1979]. Records are artificial contructs which often do not correspond to any real-world object, and tend to be difficult to design. The problem with record-based systems such as the three conventional database models described, is that they are lacking in the notion of an abstract object [McLeod 1980]. In general it appears that relational systems suffer from fewer disadvantages than the others, and have the benefits of definiteness and mathematical rigour. However, as with network and hierarchical systems, their implementation by a particular DBMS plays an important role in determining their true effectiveness. A more detailed description of these three types of database system has been provided in Appendix A. DMS 1100, the network database system on and for which ADD was developed, will be described in the next section.

2.4. The DMS 1100 Database System.

The Codasyl Data Base Task Group (DBTG) of 1971 developed the guidelines for network database systems, specifying the syntax and semantics of the command language or DML (Data Manipulation Language) and of the DDL (Data Definition Language) in which a schema is written [Codasyl 1971]. At subsequent meetings of the DBTG, extensions and improvements were incorporated, such as the specification of an SDDL (Subschema Data Definition Language). The DBTG proposals have received a great deal of attention and although their merit is still a fairly contentious issue, nevertheless several DBMS's have been implemented based on these guidelines. DMS 1100, the database system on the Univac 1108 at UCT, is one such example which adheres fairly strictly to the DBTG specifications.

A DMS 1100 schema consists of five sections. The first (data names) and last (table section) are only used in special circumstances, and will not be discussed here. (Appendix B gives the complete syntax of the DMS 1100 DDL.) The second section of a schema names and describes the files, called AREAS, on which the database will reside, giving information such as the number and size of their pages (the unit of access), designating overflow pages, etc.

The next section defines the record types, giving COBOL-like descriptions of their fields. Integrity constraints can be attached to items and/or records by using CHECK and RESULT clauses. The "location mode" (storage/access strategy) must be specified for each record type. This can be DIRECT (program supplies physical address of record); CALC (stored by a hashing technique); INDEX SEQUENTIAL or VIA SET. The latter implies a record is stored as near as possible to the owner ("parent") of the specified set.

The remaining section describes the sets in the database. A set represents a 1:many relationship between an owner record and several member records (possibly of different types). For example set SUPPLIERPARTS can have owner SUPPLIER and member PART. Membership can be automatic(compulsory) or manual(optional). The retention clause suggested by the DBTG is not implemented in DMS 1100. Each set has, amongst others, two features denoting ordering of members ("sorted" by designated keys; "first", "last", etc.) and SET SELECTION (which governs the choice of a specific set occurrence). Appendix H shows an example of a DMS 1100 schema.

To access a DMS 1100 database, three forms of DML are provided,

for embedding in COBOL, FORTRAN and PL/1 programs respectively. The FORTRAN DML which was used in the ADD system adheres very strictly to that laid down by the DBTG, and consists of commands such as STORE, MODIFY, FIND, ERASE, DISCONNECT etc. Appendix C gives the syntax of the ASCII FORTRAN DML supported by DMS 1100.

2.5. Normative Theory.

The major research area in database technology today is centred around the concept of normalisation and normal form theory. Normative theory is an attempt at creating rules for "good" relational databases, but can also be applied to obtain "good" record structures for hierarchies or networks. The significance of normal form theory lies in the fact that it is based on a sound mathematical foundation and is a "first step" towards making database design an exact science rather than an ad-hoc process. In this section the concepts on which this theory is based, are defined. Thereafter several Normal Forms are described.

2.5.1. Definitions.

A set of attributes X of a relation R is said to be functionally dependent on a set of attributes Y in R, if every set of values for Y is associated with a unique set of values for X. This is denoted Y -> X and is read as "Y functionally determines X". If there is no subset S of Y for which S -> X, then we say that X is fully functionally dependent on Y. Intuitively, this means there are no "unnecessary" attributes in Y. A candidate key for a relation is a set of attributes in the relation which fully

functionally determine all attributes of that relation. A prime attribute is any attribute that is a part of some candidate key for R, and all other attributes are termed "non-prime". If a relation R has several candidate keys, one is chosen as the 'main' key, called the primary key. There are several axioms relating to functional dependencies (FDs), such as the transitivity axiom [Bernstein 1976]. This states that if X -> Y and Y -> Z , then X -> Z by transitivity, or "Z is transitively dependent on X".

Recently, the more general notion of a multivalued dependency (MVD or MD) was discovered by Zaniolo[Zaniolo 1976] and by Fagin [Fagin 1977]. A multivalued dependency is written X ->-> Y, and is read as "X multi-determines Y". To illustrate this let us denote the remaining attributes of the relation, i.e. R - (X union Y), as Z. The MVD implies that whenever the tuples (x1,y1,z1) and (x1,y2,z2) appear in the relation, tuples (x1,y1,z2) and (x1,y2,z1) also exist. This shows that the relationship between X and Y is in no way dependent on the other attributes in R. As an example, if R consisted of three attributes EMPLOYEE, CHILD and PROJECT then there is a multivalued dependency of CHILD on EMPLOYEE, since the child of an employee is independent of the projects on which he is working. Thus if we have tuples <Jones,Bob,P1> and <Jones,Jill,P2>, then we expect to also find tuples <Jones,Bob,P2> and <Jones,Jill,P1>. An FD is clearly a special case of an MVD.

Aho, Beeri and Ullman [Aho 1979] recognised another type of dependency that can occur in a relation, called a join dependency. A relation R satisfies a join dependency *(X,Y,...,Z) if and only if it is equal to the join of its projections on X,Y,... ,Z; where X,Y,...,Z are sets of attributes of R. As an example, consider the relation R composed of attributes STUDENT, COURSE and PROJECT.

This has <STUDENT,COURSE,PROJECT> as its key, if several STUDENTS from different COURSES can be involved in the same project. R satisfies the join dependency (JD) *((STUDENT,COURSE), (COURSE,PROJECT), (STUDENT,PROJECT)) because if tuples (Smit,C3,P6), (Jones,C3,P1) and (Smit,C9,P1) appear in R then (Smit,C3,P1) must exist in R. Fig. 2.1 shows that R is the join of its projections on (STUDENT,COURSE), (COURSE,PROJECT) and (STUDENT,PROJECT). Zaniolo recently introduced several new concepts, such as elementary FDs and multiple elementary FDs. An elementary FD X -> Y is one where Y is a single attribute and there is no subset S of X for which S -> Y. An elementary MVD X ->-> Y is an MVD such that there is no S contained in X, T contained in Y, for which S ->-> T. A multiple elementary MVD is an elementary MVD for which there exists another elementary MVD with the same left hand side. Zaniolo and Melkanoff have shown that All the FDs and MVDs of a relation are inferable from its elementary FDs and multiple elementary MVDs [Zaniolo 1981]. An elementary key attribute for a relation R is an attribute which is part of a key X such that some X-> A is an elementary FD of R.

Admissibility is another interesting concept. A set of elementary FDs Z is admissible with respect to a collection of atomic relations A (i.e. relations with no non-trivial multiple elementary MVDs) if and only if the following conditions hold: (1) if Z contains an elementary FD with scope W (i.e. W is the union of the attributes on the left and right hand sides), it must contain every other elementary FD with scope W; and if $\Pi_R(W)$ is atomic, then A must contain it. (2) if A contains an atomic projection $\Pi_R(W)$, Z must contain every elementary FD of R having scope W.

STUDENT	COURSE	PROJECT
Smit	C3	P6
Smit	C9	P1
Jones	C3	P1
Smit	C3	P1

STUDENT	COURSE	COURSE	PROJECT	STUDENT	PROJECT
Smit	C3	C3	P6	Smit	P6
Smit	C9	C9	P1	Smit	P1
Jones	C3	C3	P1	Jones	P1

JOIN

STUDENT	COURSE	PROJECT
Smit	C3	P6
Smit	C3	P1
Smit	C9	P1
Jones	C3	P6
Jones	C3	P1

JOIN

STUDENT	COURSE	PROJECT
Smit	C3	P6
Smit	C3	P1
Smit	C9	P1
Jones	C3	P1

Figure 2.1 An Example of a Join Dependency

2.5.2. Normal Forms.

Several design rules or 'normal forms' have been defined for converting a given relation scheme into a 'better' collection of relations. The resultant scheme is considered better in that it decomposes a given relation to remove embedded relationships; in this way it prevents a number of update problems from arising.

A relation is in 1st Normal Form (1NF) if every attribute in the relation is atomic. This means that a 1NF relation has no data aggregates or repeating groups. Such a relation is said to be normalised.

A 1NF relation is in 2nd Normal Form (2NF) if each of its non-prime attributes is fully functionally dependent on the primary key of the relation [Codd 1970]. For example a relation R with attributes STUDENT, COURSE, FINALMARK and LECTURE-PERIOD is not in 2NF. Its key is <STUDENT,COURSE> but COURSE -> LECTURE-PERIOD. This is a separate relationship in its own right and hence should form a separate relation R2. If R is replaced by two 2NF relations R1 (STUDENT,COURSE,FINALMARK) and R2 (COURSE,LECTURE-PERIOD) then several update anomalies disappear because of the fact that the period in which a course is taught is now stored once only. This saves space, and facilitates modification of the database if the LECTURE-PERIOD of a course is changed. If R is retained, a situation could arise where some tuples for a course have their LECTURE-PERIOD altered, but not others, resulting in an inconsistent database. Using R1 and R2 this cannot occur. These two relations can also increase the information content of the database as a tuple can be inserted into R2 for a new course and its alloted period, before any

students register for that course. Similarly if all the R2 tuples for a course are deleted, the period associated with it is retained in an R2 tuple. Thus there are clearly advantages to decomposing a relation such as R into several 2NF relations.

A relation is said to be in 3rd Normal Form (3NF) if it is in 2NF and none of its non-prime attributes is transitively dependent on a key of the relation. This eliminates FD's between non-prime attributes. since they represent a separate relationship they should constitute a separate relation. For example if relation R has attributes PROJECT, DATE, DESCRIPTION, MANAGER and MANAGER-TELEPHONE with key PROJECT, then MANAGER-TELEPHONE is transitively dependent on PROJECT. If this were replaced by two relations R1(PROJECT,DATE,DESCRIPTION,MANAGER) and R2(MANAGER, MANAGER-TELEPHONE) then each telephone-number would be stored once only, instead of once for each project involved. The insertion, deletion and update anomalies would thus disappear, in a similar way to when a 1NF relation is replaced by 2NF relations.

Boyce-Codd Normal Form (BCNF) takes this idea to its natural conclusion. Suppose relation R has attributes S#,P#,SNAME and QUANTITY, and the following FDs hold: S#,P# -> QUANTITY, S# -> SNAME and SNAME -> S#. There are two overlapping candidate keys SNAME,P# and S#,P#. This relation is in 3NF, as QUANTITY is its only non-prime attribute. However, there is an embedded relationship between S# and SNAME, which should form a separate relation to prevent update anomalies arising. This type of situation led to the identification of BCNF. A relation is in BCNF if the only FDs among its attributes are those in which a candidate key functionally determines attribute(s) of the relation. Since S#->SNAME and SNAME -> S#, and niether S# nor

SNAME is a candidate key for R, R is not in BCNF.

Fourth Normal Form extends the above to incorporate MVDs, and is defined as follows. A relation is in 4NF if and only if the only MVDs among its attributes are those in which a candidate key multidetermines attribute(s) of the relation. Since an FD is a special case of an MVD, 4NF is clearly a stronger condition than BCNF.

The ultimate normal form (with respect to projection and join) is fifth normal form (5NF.) "A relation R is in fifth normal form (5NF) - also called projection-join normal form (PJ/NF) - if and only if every join dependency in R is implied by the candidate keys of R" [Date 1981]. Fagin [Fagin 1977] has proved that an MVD is a special case of a join dependency. Hence 5NF is a stronger condition than 4NF. It is interesting to note that a relation scheme can always be decomposed into an equivalent 3NF, 4NF, BCNF or 5NF scheme in a "non-loss" way. However, it is not always possible to convert a relation scheme into an equivalent BCNF, 4NF or 5NF scheme and preserve all its dependencies.

Other normal forms have recently been proposed, such as Elementary Key Normal Form (EKNF) [Zaniolo 1982b] and Improved Third Normal Form [Ling 1981]. Both enforce the separation principle more strictly than does 3NF. Zaniolo developed EKNF, because he felt that 3NF does not enforce separation sufficiently, while BCNF is prone to computational complexity and is incompatible with the principle of representation. The representation principle concerns the complete representation of dependencies in a relation scheme. This criterion is preferable to that of removing update anomalies, which according to Zaniolo is an elusive and possibly unattainable objective. A relation is in EKNF if and only if for

all elementary FDs of R, X -> A, either X is a key for R or A is an elementary key attribute for R. Zaniolo shows examples where EKNF improves on 3NF relations and proves that Bernstein's algorithm for synthesising 3NF relations [Bernstein 1976] does in fact create EKNF relations.

A relation Ri in a relational schema R is in Improved 3NF if each non-essential attribute is not restorable in Ri. An attribute is 'restorable' if it is derivable from the other relations of R, and is 'non-essential' if it is not required to derive the value of any other attribute of R. This normal form was discovered by Ling, Tompa and Kameda [Ling 1981] and involves detecting superfluous attributes, whose values are derivable using other relations. An analogous definition of Improved BCNF is: a relation Ri in a relational schema R is in Improved BCNF if each attribute is restorable in Ri. However unlike improved 3NF, such a schema may not exist.

When one considers this theory of normalisation, one realises that although the projections to higher normal forms do result in better relations in the sense that update anomalies cannot occur, they also result in very many 'simple' relations. Thus there is a definite trade-off between integrity and efficiency considerations, because if one relation is decomposed into many, then several joins may be needed to answer queries. Griffith [Griffith 1982] says a relation scheme that has been decomposed or 'factored' too much is harder to reorganise. He concludes that "as the number of different desired factorisations increases, the net benefit of factoring decreases. Information bases which must support many widely diverse or frequently changing applications may be better off not factoring at all". Whereas 1NF is generally

accepted and most authors advocate 3rd Normal Form relation schemes, 4th and especially 5th Normal Form are somewhat contentious. Although it is clear that MVDs and join dependencies indicate embedded relationships, the question is whether or not they should be removed to form separate relations. Several database design methodologies generate 4NF schemes and then join relations to meet performance requirements, returning to a 3NF or even a 1NF scheme. Opinion on normal form theory is best summed up by these words: "While everyone agrees that 'Small is good', the issue 'How small is good enough?' is somewhat controversial" [Zaniolo 1982b].

2.6. Some Database Design Methodologies.

Many database design methodologies approach the problem at a very high level and do not give sufficient detail as to how to perform each of the steps they advocate. Nevertheless these are useful when determining the main stages in database definition. Hence some of these are presented before considering more specific, detailed methodologies.

A database design process should consists of four phases: data gathering and requirements analysis, constructing a model of the application, converting this to a logical database structure and finally adding physical parameters [Ledgard 1977]. Sundgren [Sundgren 1978], amongst others [Tsichritzis 1978], emphasises the use of prototypes and tuning during the database design. Using his method the external requirements are formulated as a goal function, which is tested for consistency and feasibility. After arriving at a logical design and transforming this to be

compatible with the DBMS, this is checked to see if it satisfies the goal function. The process is repeated until the goal is 'met'. This view is also held by others [Appleton 1980], who centre their methodology around the installation of a prototype. This, supplemented by a data dictionary, is subjected to heuristic analysis and environment tests. The database is modified and the process repeated until a satisfactory design is achieved.

Teorey and Fry [Teorey 1979] build their design methodology around the processing requirements of the organisation. The data items as well as their data volume and processing frequency are considered. They question the idea of normalisation, because of the trade-offs between integrity and efficiency. They design an information structure diagram (ISD) and then draw a subset of this for each application, consolidating these to get a revised ISD. This is refined using quantitative information and performance measures. Physical parameters are then chosen, followed by the inclusion of integrity and security constraints. In contrast Bubenko [Bubenko 1977], who also bases her design on requirements expressed in terms of queries and transactions, follows this with the identification of entities and the specification of FD's between them. The next step, information analysis, includes possible re-iteration of this process, before finally introducing names and mapping to a DBMS structure. Kahn , on the other hand, uses a name-based model from the beginning [Kahn 1976]. Her methodology caters for a community of users each responsible for a specification, and thus begins by consolidating these into a global information structure. Once again, entities are determined and the FD's between them used to generate normalised relations. This structure is refined by introducing security constraints and controlled redundancy to ensure adequate preformance, with the

final step being DBMS accomodation.

Smith and Smith [Smith 1977] advocate using a model consisting of data aggregates from which one can proceed bottom-up or top-down to obtain other aggregates. Aggregation, which will be discussed in the next chapter, is the abstraction whereby an object is constructed from its constituent objects. The aggregates are converted to 4th Normal Form and then keys are established. A slight variation of this methodology is proposed by Jefferson [Jefferson 1980], who lists the design stages as (1)creation of SADT (Structured Analysis and Design Technique) diagrams from a requirements statement; (2) developing different levels of aggregation; (3) determining logical records from their aggregates and (4) designing the physical database structure.

In addition to these high-level design guides, there are some more specific methodologies which also explicitly state ways of obtaining the logical structure. The major difficulty lies in the conversion of a collection of entities, attributes and associations into records or relations. This particular problem is addressed by Taylor and Frank [Taylor 1976] who provide guidelines for grouping attributes 'correctly'. They advocate grouping together all keys for an entity and including with these all attributes X such that X describes the entity and the entity key functionally determines X, unless X is part of a key of another entity. They also suggest that repeating attributes should be considered to decide whether they are worthy of being represented as a separate entity or not. However, they do not suggest how this decision should be taken.

Wang and Wedekind [Wang 1975] have developed a complete methodology based on normative theory. This commences by

collecting the complete set of attributes and functional dependencies, computing a minimal cover and then determining an optimal 3NF relation scheme based on this. Relations may subsequently be combined to form records according to performance requirements. This is usually necessary because their methodology tends to generate a separate relation for each simple fact; and they actually suggest a feasible trade-off is to join relations, obtaining a 1NF scheme! The algorithm to perform the conversion to a set of 3NF relations was developed by Bernstein [Bernstein 1976], but it relies on a specification of all FDs between items. This is the main disadvantage of this method [Buchman 1979]. Another problem, first identified by Bernstein himself, is that "under the synthetic approach every conclusion is tentative and subordinate to a semantic validity check" [Zaniolo 1981]. For example, if the given FDs are f1:DEPT -> SECTION; f2: SECTION -> MANAGER and f3: DEPT -> MANAGER, then relation synthesis algorithms will ignore f3, as it appears to be inferable from f1 and f2 by transitivity. This will be incorrect if f2 means each SECTION has one MANAGER and f3 implies each DEPT has one MANAGER. Another peculiarity of relation schemes synthesised by Bernstein'S algorithm is that these can sometimes contain relations R1 and R2 such that R1 is a projection of R2 [Zaniolo 1981].

Ling, Tompa and Kameda [Ling 1981] have devised an algorithm for generating Improved 3NF schemes based on Bernstein's preparatory algorithm. This uses a superfluous attribute detection algorithm to convert the relations generated by the preparatory algorithm to IMproved 3NF.

Zaniolo and Melkanof carry the decomposition further than 3NF. In contrast to the synthetic approach of Bernstein and of Wang and

Wedekind, they commence their design with a given relation and a sample database content. This is then decomposed. Their opinion is that with smaller units as building blocks, greater data independence can be achieved. Thus they state that "the presence of a non-trivial MD (MVD) is sufficient and necessary for a relation to be composed into a pair of proper subprojections, without losing information on the content of the relation". Their decomposition algorithm produces a pair (ACOVER,ZCOVER). ACOVER is a collection of atomic relations. ZCOVER is a set of elementary FDS constituting a minimal cover which is admissible with respect to ACOVER. The resulting relation scheme not only has minimal redundancy, it also ensures complete relatibility (complete representation of the dependencies that apply to the data.) However, there exist relations for which this decompostion will fail, requiring intervention by the designer [Zaniolo 1982a]. Furthermore, as with the algorithms of Bernstein and Wang and Wedekind, the quality of the schema depends on the correctness and completeness of the original dependencies. For example, latent dependencies must also be given initially. These are dependencies not deducible from the original relation but inferable from one of its projections. Zaniolo and Melkanoff have shown how the synthesised relation scheme can be converted into graphs, and suggest that these graphs can be used to design network or hierarchical databases. The first step in this design process would be combination of atributes to form entities [Zaniolo 1982a]. No guidelines for performing this step are given, however.

Functional dependencies also play a part in Shen's methodology [Shen 1978]. This generates a relation scheme consisting of a relation for each: repeating attribute, entity(object) and its non-repeating attributes, association and its non-repeating

attributes, and association without attributes. The latter are decomposed into equivalent atomic relationships first - that is, if X -> Y, X -> Z and Y does not determine X then relation (X,Y,Z) is split into 2 relations (X,Y) and (X,Z). Redundant associations are removed and then the result converted to a set of optimal 4NF relations.

The methodology of Chang and Cheng [Chang 1978] is based on a data model which is set up as a graph comprising entities linked by functional dependencies. An efficient algorithm is used to derive a minimal cover graph with no redundant FD's. From this the records, items and sets are designed. Records are formed by grouping with each attribute all those attributes which it functionally determines. If two records Ri and Rj are key compatible (i.e. their keys are drawn from the same domain(s)) and key Ki -> Kj, then set Sji is formed.

Grabowski and Eigner [Grabowski 1979] use Schmid and Swenson's data model [Schmid 1975] as a basis for their design philosophy. Unlike the previous methods, functional dependencies play no part in this design. Each entity becomes a separate relation consisting of all its attributes, excluding repeating attributes and "complex" attributes (i.e. having their own key). Separate relations are created for such attributes. Data aggregates are separated into their constituent attributes wherever they appear. Each association also becomes a relation and the integrity rules are converted to domain- and tuple-oriented, and intra- and inter-relational.

All the abovementioned methodologies do little more than outline the database design process. There is also very little aid for designers in the form of automated tools. However, some such

tools have been developed independently of any design philosophy. Tompa developed a system to choose an efficient physical representation for a given structure. It uses a table with one element for each potential representation of each data type, storing expected time and storage requirements. As all possibilities cannot be compared, he uses a branch and bound method [Bernstein 1976] to choose a 'minimal cost' schema.

Hammer [Hammer 1977] has also created an automated system for choosing an efficient physical database design. However, he stresses that it should be viewed strictly as a tool to guide the Database Administrator, who must judge whether the solution presented is plausible, and modify it manually where necessary. A small subset of possible structures (chosen by heuristic methods) is submitted to an evaluator. Changes are made and the structures resubmitted to see if the figure of merit improves or not, until no more improvements can be found. Other attempts use a closed-form analytic expression for the cost of a structure, parameterized in terms of key properties of the design, and subject this to mathematical optimisation. However such methods have been critisized for their oversimplification of the problem [Hammer 1977].

CSDL [Roussopoulos 1979], the Conceptual Schema Definition Language, is a "design tool with which the designer can express his understanding of the application". This deals with information gathering and data model creation and is an aid to incremental development of a conceptual schema. Based on the semantic network model, the system constructs and displays a graphical representation of user's input, which can be modified with a light pen for greater clarity. The structure can be

edited or deleted, and is checked for consistency by the system which includes diagnositc and trace facilities.

In the literature only two systems were found which automate the entire process of obtaining a logical, DBMS-compatible database structure from an initial requirements specification. Since these have essentially the same objectives as ADD, each is now described in some detail. The first generates a hierarchical database design given as input a binary relationship model [Hubbard 1979]. That is, each relationship involves only two entities and is either simple ("to one"), or complex("to many") or conditional. A conditional mapping is "to one", but may have no target entity. Where the inverse relation is unspecified, a complex one is assumed.

The first phase removes transitivity in the input. An attribute is taken to be anything which is the target of a relationship, and a key is anything that is the source of one. Each key together with its dependent attributes constitutes one segment. Complex relationships between keys derive a parent-child link between the corresponding segments, unless the inverse is also complex, in which case a separate tree type is generated. Conditional associations are treated as simple, if their target is not a key, with the target flagged as optional. Otherwise they are dealt with as for complex relationships, to avoid having segments that may not have a parent in the database. The system detects loops and branches of more than 15 levels (not permitted by the DBMS), but does not resolve these problems.

The importance of human intervention is considered essential and numerous lists are made for consideration by the Database Administrator as candidates for modification. These include

repeating attributes; relationships removed because of transitivity; many-to-many associations for which new tree types were defined (together with any segment that could be used for virtual pairing); keys related one-one, e.g. NAME and IDENTITY-NUMBER; key-only segments and any performance weight calculations. The latter is an optional part of the system, used only if input includes performance-oriented information. This chooses the 'best' parent from several candidate parents according to the weight of the corresponding paths.

As is pointed out "due caution should be exercised by the designer, however, for these weights are only as good as the estimates from which they are calculated" [Hubbard 1979]. In addition to the problem areas highlighted by the listings, other alterations may be desirable, such as splitting or joining segments. Thus the system can be seen to have definite limitations and to be rather simplistic, being based on a somewhat rudimentary collection of facts. It ignores integrity, security and physical considerations. However, it does generate a protoype which together with the auxiliary listings, should be of some value to a database designer. A suggested method for converting the same input to a relation scheme is outlined but has not been automated.

The other automated design system was developed as a Ph.D thesis by Gerritsen [Gerritsen 1975], who defines his database structure on functional requirements. That is, his 'DESIGNER' system is based solely on a set of anticipated queries. It is part of a system which also generates COBOL/DML routines for these queries. The queries are submitted in a special language called HI-IQ, with prompts from the system and indentation of replies assisting the

user to phrase these. All names used in these queries, eg. SHIPS and CAPTAINS, must be submitted initially, and then any other names encountered are treated as errors.

By examining the queries the DESIGNER derives a set of assertions about the application. These are then processed and record inter-relationships constructed accordingly, followed by record contents(fields) and key designation. Initially only three types of assertion are made : ABOVE(A,B) to indicate record A must be above record B; INORABOVE(I,A,B), meaning item I must be in or above record A, unless records A and B exist in a confluent hierarchy (i.e. two paths containing A and B respectively, 'meet' at a common member record); and CALCPORT(I) to denote that the value of item I was tested in a query and hence it is a 'good' candidate for the key of some record. All these assertions are derived according to the context in which the names I, A and B appear in a query.

The assertion processing phase involves the following steps. FIrstly CONFLUENCY(A,B) is created wherever the pair ABOVE(A,B) and ABOVE(B,A) is detected. Thereafter, redundant ABOVE assertions are eliminated so that confluent hierarchies can be constructed. This is done when assertions CONFLUENCY(A,B), ABOVE(A,C) and ABOVE(B,C) are encountered. If no record such as C can be found to serve as 'Meeting point' for a confluency, a system record is generated for this purpose.

Once a network has been constructed from the derived set of assertions, the INORABOVE assertions are processed in a similar manner to obtain the fields of the records. Firstly, redundant INORABOVE assertions are erased. Any item I that remains in only one INORABOVE(I,A,-) assertion is then placed in record A. If an

item still exists in more thanone INORABOVE assertion, then a record is found which connects all these substructures, and I is placed there. For example, if we have INORABOVE(I,A,-), INORABOVE(I,B,-) ABOVE(C,A) and ABOVE(C,B), then I becomes part of record C. If no record connects these substructures, a system record is created for this purpose.

This system is intuitively appealing because of its simplicity, but this is also its major shortcoming. Unless a comprehensive set of queries, correctly and consistently phrased, is used for the design, an unacceptable structure will result. The philosophy of basing a designer on functional requirements rather than on the data characteristics and information perspective, is questionable. Not only do functional requirements change in time - and the initial structure may be incapable of dealing with new queries, but it is also more difficult to specify a complete set of envisaged queries than it is to describe an organisation and its data. The DESIGNER is limited to logical structure creation, ignoring all physical problems and integrity and security issues. The system takes no account of "data characteristics such as relative volumes, volatility of data, etc.... Ignoring these design issues and data characteristics will never result in incorrect data structures, but it may lead to the design of inefficient structures" [Gerritsen 1975]. The DESIGNER also cannot recognise recursive relationships, which may not exist in CODASYL databases.

From this brief resume, it should be evident that no complete, universally-accepted database design methodology exists for the entire process from the information gathering to the creation of a schema compatible with some DBMS. Methods of converting relation

schemes to "better" relation schemes and of creating relational, network or hierarchical structures given a data model of the application, have been developed. However these rely a great deal on the designer's intuition. In the words of Wang and Wedekind: "Methods used in the design of database systems are essentially trial-and-error, supported by neither a scientific foundation nor an engineering discipline" [Wang 1975]. Bubenko and Yao purport that "Research efforts on design methodologies seem to have generated few solutions to practical, real-life problems. At the current state-of-the-art, most practical methods in the area of database design are ad-hoc approaches... there seems to exist a considerable gap between the researchers who work on theoretical aspects of database design and the practitioners who deal with complex, real-life problems" [Bubenko 1979].

2.7. Database Design Techniques.

Several precepts for good database design exist, some fairly well-established and others still rather contentious. Most authors advocate simple and standard solutions rather than "too smart" designs [Sundgren 1978, Ledgard 1977]. A systematic and well-structured approach to the design process is considered essential, with adequate mapping facilities for smooth transitions from one stage to the next. The requirements specification should be analysed for errors and inconsistencies to detect a false perception of the real world. A data dictionary is generally agreed upon as being helpful in the early design stages. Wasserman [Wasserman 1978] stresses that it should not be too difficult to incorporate changes in the data organisation or access privileges. This latter criterion is also mentioned as a

design goal by Ledgard and Taylor [Ledgard 1977]. It is generally recognised that performance oriented considerations should be introduced separately [Wang 1975, Hammer 1977, Kahn 1976, Buchman 1979, Appleton 1980] to ensure a flexible database reflecting the nature of the real world, rather than one too tightly bound to current applications.

There are however several aspects of database design in which opposing ideas are held by researchers. For example, some authors advocate a 3NF or 4NF structure, while others favour controlled redundancy for greater efficiency and flexibility [Grabowski 1979] or argue that items describing a record should not be stored in another record [Senko et al 1973]. Many methodologies obtain a 3NF design and introduce redundancy thereafter.

Palmer [Palmer 1978] is wary of basing a design on normative theory and says "Database design is a practical task. It is the pragmatic methodologies rather than the purist methodologies which are being applied in database installations".

The starting point is equally difficult to define, with most methodologies first distinguishing the entities and then their properties, and others giving the totality of attributes and then determining entities as common domains of certain attribute groups. A related issue is whether to proceed "top-down" or "bottom-up". The latter has the disadvantage of immediately immersing the designer in the mass of detail at the bottom level. The top-down approach is less safe however, as it begins with a preconceived idea of the application as a whole, which could be biased, with misconceptions and/or omissions.

The question of how to integrate the information and processing

perspectives has also not been agreed upon. The latter can be introduced gradually to refine the schema. Alternatively a separate, processing-oriented schema can be designed and then the two amalgamated. There is also no concensus on when to introduce security constraints. For example, Kent [Kent 1977] advocates developing two separate schema, one to describe the enterprise, the other with integrity constraints and access authorisation. Procedure are then needed to enforce consistency between them. In contrast, [Buchman 1979] advises incorporation of security in the initial specification, because it could greatly influence the conceptual schema produced.

A very basic point on which there is some doubt is whether portions of different methodologies can be integrated. Some feel that this is not advisable if the methodologies differ in their perception of the role of an item, while others such as [Buchman 1979] believe it to be the best way of arriving at a universally-accepted methodology.

The state-of-the-art of database design is probably best summed up by these words of Sundgren [Sundgren 1978] "to some extent the literature on database design still has the character of a Swedish 'smorgasbord' exhibiting an impressive menu of tempting methods and techniques. This has to be remedied by integrating the existing and emerging pieces of knowledge into a common framework". The feeling amongst researchers is that a sound database design methodology, that creates schemas which can be proven to satisfy stated requirements, is within reach.

[Aho 1979] Aho, A.V., Beeri, C. and Ullman, J.D. "The Theory of Joins in Relational Databases", ACM Trans. on Database Systems Vol. 4, 1979, pp. 297-314.

[Appleton 1980] Appleton, D.S. "Implementing Data Management", AFIPS 1980 NCC Conf. Proc., pp. 307-316.

[Bernstein 1976] Bernstein, P.A. "Synthesizing Third Normal Form Relations from Functional Dependencies", ACM Trans. on Database Systems Vol. 1, 1976, pp. 277-298.

[Bradley 1982] Bradley, J. "File and Data Base Techniques", Holt, Rinehart and Winston Publishers, New York NY, 1982.

[Bubenko 1977] Bubenko, J. A. Jr., "Validity and Verification Aspects of Information Modeling", Proc. 3rd Int. Conf. on Very Large Databases, 1977, pp. 556-565.

[Bubenko 1979] Bubenko, J.A.Jr. and Yao, S.B. "Database Design Tools", in Issues in Database Management, Proc. of the 4th Int. Conf. on Very Large Databases, (ed.s Weber,H. and Wasserman, A.I.), North-Holland, 1979, pp. 25-26.

[Buchman 1979] Buchman, A.P. and Dale, A.G. "Evaluation Criteria for Logical Database Design Methodologies", Computer-Aided Design, Vol. 11, No. 3, May 1979, pp. 121-126.

[Chang 1978] Chang, S.K. and Cheng, W.H. "Database SKeleton and its Applications to Logical Database Synthesis", IEEE Trans. on Software Engineering, Vol. SE-4, January 1978, pp. 18-30.

[Codasyl 1971] Codasyl Committee on Data System Languages, Codasyl Database Task Group Report, ACM New York NY, 1971.

[Codd 1970] Codd, E.F. "A Relational MOdel for Large Shared Databanks", Comm. of the ACM, Vol. 13, No. 6, June 1970, pp. 377-387.

[Date 1981] Date, C.J. "An Introduction to Database Systems", (3rd ed.), Addison-Wesley, 1981.

[Deen 1977] Deen, S.M. "Fundamentals of Database Systems", MacMillan Press, London, 1977.

[Fagin 1977] Fagin, R. "A Normal Form for Relational Databases that is Based on Domains and Keys", Tech. Report Rj2520, IBM Res. Laboratory, San Jose CA, 1979.

[Gerritsen 1975] Gerritsen, R. "A Preliminary System for the Design of DBTG Data Structures", CACM Vol. 18, 1975, pp.551-557.

[Grabowski 1979] Grabowski, H. and EIgner, M. "Semantic Datamodel Requirements and Realization with a Relational Datastructure", Computer-Aided Design, Vol. 11, No. 3, May 1979, pp. 158-167.

[Griffith 1982] Griffith, R.L. "Three Principles of Representation

for Semantic Networks", ACM Trans. on Database Systems, Vol. 7, No. 3, September 1982, pp. 417-422.

[Hammer 1977] Hammer, M. "Self-Adaptive Automatic Database Design", AFIPS 1977, pg. 123.

[Hubbard 1979] Hubbard, G.U. "Computer-Assisted Logical Database Design", Computer-Aided Design, Vol. 11, No. 3, May 1979, pp. 169-179.

[Jefferson 1980] Jefferson, D.K. "The Development and Application of Data Base Design Tools and Methodology", Proc. of the 6th Int. Conf. on Very Large Databases, Montreal, October 1980.

[Jankowitz 1982] Jankowitz, H., Kilfoil, P.W. and Rabkin, I. "DBTRACE: A High-Level Database Performance Monitoring and Debugging System", Technical Report, University of Cape Town, 1982.

[Kahn 1976] Kahn, B.K. "A Method for Describing Information Required by the Database Design Process", Proc. ACM SIGMOD, 1976, pp. 53-64.

[Kent 1979] Kent W. "Limitations of Record-Based Information Models", ACM Trans. on Database Systems, Vol. 4, No. 1, March 1979, pp. 107-131.

[Kent 1977] Kent, W. "New Criteria for the Conceptual Model", in Systems for Large Databases, (eds. Lockemann and Neuhold), North-Holland, 1977, Proceedings of the 2nd Conf. on Very Large Databases, Belgium, 1976.

[Kroenke 1977] Kroenke, D. "Database Processing: Fundamentals, Modeling, Applications", Science Res. Associates, Palo Alto CA, 1977.

[Ledgard 1977] Ledgard, H.F. and Taylor, R.W. "Two Views of Data Abstraction", CACM Vol. 20, No. 6, 1977, pp. 382-384.

[Ling 1981] Ling, T.-W., Tompa, F.W. and Kameda, T. "An Improved Third Normal Form for Relational Databases", ACM Trans. on Database Systems, Vol. 6, No. 2, June 1981, pp. 329-346.

[Martin 1975] Martin, J. "Computer Data-Base Organisation", Prentice Hall, Englewood Cliffs NJ, 1973.

[McLeod 1980] McLeod, D. "On Conceptual Database Modelling", Proc. of Workshop on Data Abstraction, Databases and Conceptual Modelling, Pingree Park Co, June 1980.

[Palmer 1978] Palmer, I. "Record Subtype Facilities in Database Systems", Proc. of the 4th Int. Conf. on Very Large Databases, West Berlin, West Germany, September, 1978.

[Roussopoulos 1979] Roussopoulos, N. "Tools for designing Conceptual Schemata of Databases", Computer-Aided Design, Vol. 11, No. 3, May 1979, pp. 119-120.

[Schmid 1975] Schmid, H.A. and Swenson, J.R. "On the Semantics of

the Relational Data Model", Proc. of ACM SIGMOD, 1975, pp. 211-223.

[Senko 1973] Senko, M.E. "Data Structures and Data Acessing in Database Systems", IBM Systems Jnl. Vol. 12, 1973, pp. 30-93.

[Shen 1978] Shen, S.N.T. "A Semantic Approach in Designing Relational Databases", ACM 1978 Annual Conf. Proc., pp. 596-601.

[Smith 1977] Smith, J. M. and Smith, D. C. P. "Database Abstractions : Aggregation", CACM, Vol. 20, No. 6, pp. 405 - 413, June 1977.

[Sundgren 1978] Sundgren, B. "Data Base Design in Theory and Practice: Towards an Integrated Methodology", Proc. 4th Int. Conf. on Very Large Databases, 1978, pp. 3-16.

[Sperry 1972] Sperry Univac, "Data Management System (DMS 1100)", UP 7909, Rev. 3A, St. Paul Minn., 1972.

[Taylor 1976] Taylor, R.W. and Frank, R.L. "Codasyl Database Management Systems", ACM Computing Surveys, Vol. 8, No. 1, March 1976, pp. 67-104.

[Teorey 1979] Teorey, T.J. and Fry, J.P. "The Logical Record Acccess Approach to Database Design", Proc. of Int. Conf. on the Entity-Relationship Approach to Systems Analysis and Design, Los Angeles CA, December, 1979.

[Tsichritzis 1982] Tsichritzis, D.C. and Lochovsky, H. "Data Models", Prentice Hall, Englewood Cliffs NJ, 1982.

[Wang 1975] Wang, C. and Wedekind, H. "Segment Synthesis in Logical Data Base Design", IBM Jnl. of Res. and Development Vol. 19, January 1975, pp. 71-77.

[Wasserman 1978] Wasserman, A.I. "A Software Engineering Approach to Database Management", in Database Management (eds. Weber, H. and. Wasserman, A.I.), North-Holland, Proc. 4th Conf. on Very Large Databases, 1978.

[Zaniolo 1981] Zaniolo, C. and Melkanoff, M.A. "On the Design of Relational Database Schemata", ACM Trans. on Database Systems. Vol. 6, 1981, pp. 1-47.

[Zaniolo 1982a] Zaniolo, C. "A New Normal Form for the Design of Relational Database Schemata", ACM Trans. on Database Systems, Vol. 7, No. 3, September 1982, pp. 489-499.

[Zaniolo 1982b] Zaniolo, C. and Melkanoff, M.A. "A Formal Approach to the Definition and the Design of Conceptual Schemata for Database Systems", ACM Trans. on Database Systems, Vol. 7, No. 1, March 1982, pp. 24-59.

3. CHAPTER 3. DATA MODELS.

One of the techniques of database design is to commence the process with a "reality model" of the application. This chapter examines the need for a good model on which to base a schema generation, and considers the issue of user involvement in model creation. Existing data models are then described and evaluated.

3.1. The Need for Data Modelling and User Participation.

The importance of a data model in designing a database has been emphasised by authors such as Wasserman [Wasserman 1978], Bourne [Bourne 1979], Sundgren [Sundgren 1978], Chang and Ke [Chang 1978b], Buchman [Buchman 1979], Grabowski and Eigner [Grabowski 1979] and Trauboth [Trauboth 1978]. It is useful as an aid to recognising objects and relationships in the application; and provides a system analysis understandable to the Database Administrator and users alike. It is especially important because of the strong effects that any misconception of the object system has on the database's ability to satisfy users. One of the main reasons why database design is such a difficult task is that there is a communication gap between designers and users. As Sundgren says "the very process of specifying the reality model will help the users and designers to understand each other's problems as they have probably never done before" [Sundgren 1978].

Sundgren is a protagonist of user participation in model creation, as are others such as Appleton [Appleton 1980]. Not only is user involvement necessary to increase users' interest, but it is also vital that they be involved with the capabilities and limitations

of the database from the outset. Sundgren goes so far as to state "We all know by now ... that user participation in the design process is essential to the success of an information system". He advocates a 'co-operative' approach, where "users and designers are prepared to accept a common responsibility for the final results of the design work". The same view is held by Lochovsky and Tsichritzis [Tsichritzis 1982], who state that the enterprise description "cannot therefore be initially in terms of a very complicated and difficult-to-understand model".

A good data model which is sufficiently simple for computer-naive users to work with, enables us to answer the question "Who should provide the system specification?" The systems analyst is unlikely to have an adequate understanding of the application environment, while the user previously could not handle the specification tools of the analyst. Now one can leave data model definition largely in the hands of users, requiring analysts only to verify consistency and possibly refine the model.

3.2. Data Models - A Survey.

"Data modelling has been one of the major themes of database research over the past ten years" [Shipman 1981]. All data models represent the real world in terms of one or more of the following basic constructs: entities, attributes, relationships, and values. However they can be broadly classified in terms of six categories, corresponding to six different ways of viewing the real world. These are entity-attribute-relationship models, binary relationship models, aggregation models, extended relational models, functional models and infological models. This section

defines the concepts of entity, attribute, relationship and value, and discusses features which can be incorporated into any model to enhance its semantic expressiveness and ease of use. Thereafter each of the six categories is described in turn, establishing their characteristic properties and analysing examples of such models.

3.2.1. Terminology.

An entity is "any thing, person, place, event or concept ... of interest to the enterprise and about which information may arise or be required" [Bourne 1979]. An attribute characterises an object or set of objects and is of interest only because it provides information about the object(s). A relationship or association is a connection between two entities, and is usually identifiable with a verb. A value is a character string, such as '369', 'Smith' or 'red', which an attribute can assume.

In recent years a number of modelling concepts have been developed to enhance the ease of use and expressiveness of a data model. A model can include several types of integrity assertion to improve its semantic capabilities. These can be applied to entities by specifying identifiers; to attributes by stipulating type, format, range, synonyms and whether or not null values are permitted, etc.; as well as by rules governing relationship formation. Associations should be described, not merely stated. For example they should be able to participate in other relationships, and should be labelled as optional or mandatory. The usefulness of associations is increased if inverses are given and role names such as IS-SUPPLIER-OF are used; the latter is referred to as a

'semantic relationship' in the literature. Indeed, the naming of elements in a model is considered to be important in ensuring that the designer fully understands his environment and gives a precise specification.

A facility which is being increasingly incorporated into data models is termed "has-subtype" [McLeod 1980] or generalisation [Smith 1977b] or IS-A hierarchy [Chang 1978]. This is the means by which an entity type can be specified as a special case, or subtype, of another entity type. For example SHIPS is a generalisation of OILTANKERS and MERCHANT-SHIPS, as these are special types of SHIPS. SHIPS can be treated as objects in their own right, with their own attributes and relationships. These also apply to OILTANKERS and MERCHANT-SHIPS which may in addition have properties and associations of their own. Generalisations can be represented by means of a tree (or "IS-A hierarchy") as shown in fig. 3.1.

The inclusion of derived information in a model enhances its scope and expressiveness, and facilitates enterprise description. Thus attributes such as TOTAL-SPILLED and AVERAGE-SIZE should be includable, with their derivation precisely defined. Three other desirable criteria are ease of use, ease of adaptation to automation and freedom of interpretation. A model is easy to use if it is simple, with no complex or confusing concepts; and is easy to learn, understand and remember. A model is unsuited to automation if it presents navigational or display problems, if it is unnecessarily large, or if verifying the consistency of a specification is difficult. freedom of interpretation is a measure of how restrictive a model is. If there is no redundancy in the model or if it is very important to 'correctly' distinguish

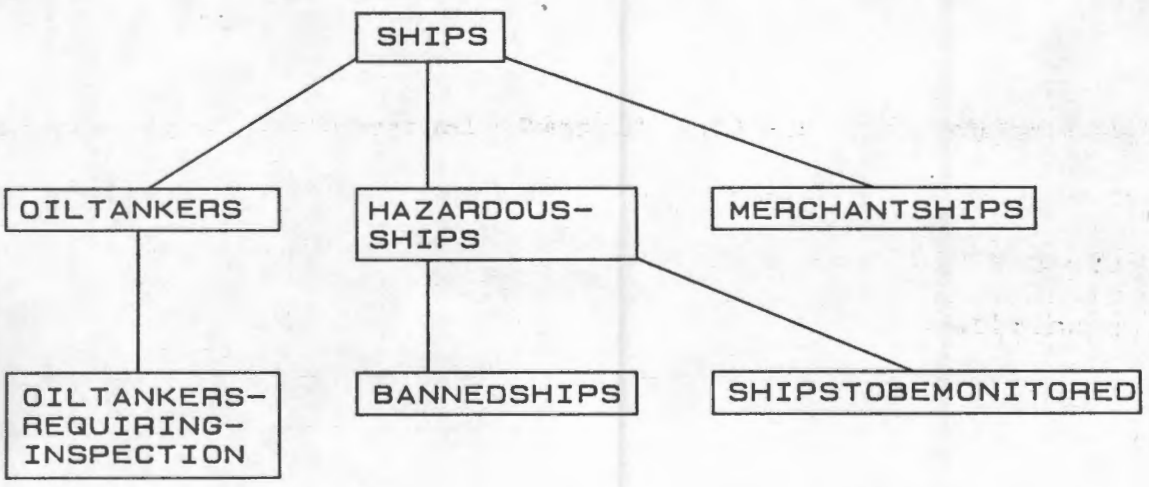


Figure 3.1 An IS-A hierarchy from a naval database.

entities from attributes and relationships, then it has a low degree of freedom. Another feature which greatly increases the value of a data model is the ability to stipulate rules for checking whether a given model is "well-defined", i.e. meaningful, or not. Some features also exist which may or may not be regarded as beneficial to a model; an example being "dynamics" constructs. Most models describe only the statics of the application, that is the data types, their meaning and interrelationshps. Another aspect of the application which could be modelled is its "dynamics" or processing requirements. "There appears to be a concensus that the information perspective should be given priority" (over processing) [Buchman 1979]. This is because "usage patterns can shift dramatically as old applications evolve or are replaced, as new applications emerge, as users acquire increased sophistication and familiarity with the system. Internal characteristics of the data may change as well, reflecting the changing nature of the domain being modelled". [Hammer 1978]. However, there has recently been a tendency amongst model designers to include descriptions of transactions. While this clearly enhances information content, it is nevertheless not a sound foundation for database design.

Incorporation of functional and multivalued dependencies, as described in the previous chapter, enables a model to be converted to 3rd or 4th Normal Form; and also increases its information content, since these are essentially special types of relationship. A problem here however is that FD's and MVD's are not easily understood or recognised by non-DP people. They require that model construction be controlled by the analyst. All these criteria will be used to evaluate the different data models as they are described.

3.2.2. Entity-attribute-relationship Models.

The entity-attribute-relationship type of model is based entirely on these three basic concepts. No structure is imposed on the model and the designer is allowed a free or unrestricted interpretation of his environment. Such models are easy to understand and use and are generally constructed in graphical format, as will be shown when considering some examples. Models in this category to be reviewed here are Chen's entity-relationship model, TAXIS, the models of Roussopoulos and of Navathe and Schkolnick, DIAL, SDM and EDSM. The most well-known of these is Chen's Entity-relationship model [Chen 1976, Chen 1977, Chen 1980]. This comprises entity sets (that is, entity types), relationship sets, attribute sets and value sets. The model can be diagrammatically represented using a graph where entities are linked according to the relationships in which they participate. (See fig. 3.2). Value sets are colours, names etc., and attributes are conceived of as mapping from entities or relationships into the Cartesian product of value sets. The role of each entity in a relationship can be named, and is labelled "to one" or "to many". Entity sets can be designated as "weak" if such entities cannot exist unless they participate in some relationship. For example, in the HOSPITAL <-> WARDS association, WARDS is a weak entity set. ID dependencies can be stated for entities which are uniquely identified by their relaitonsips to other entities. An example of this is a DIAGNOSIS entity, which must be associated with some PATIENT to be distinguishable form other DIAGNOSES.

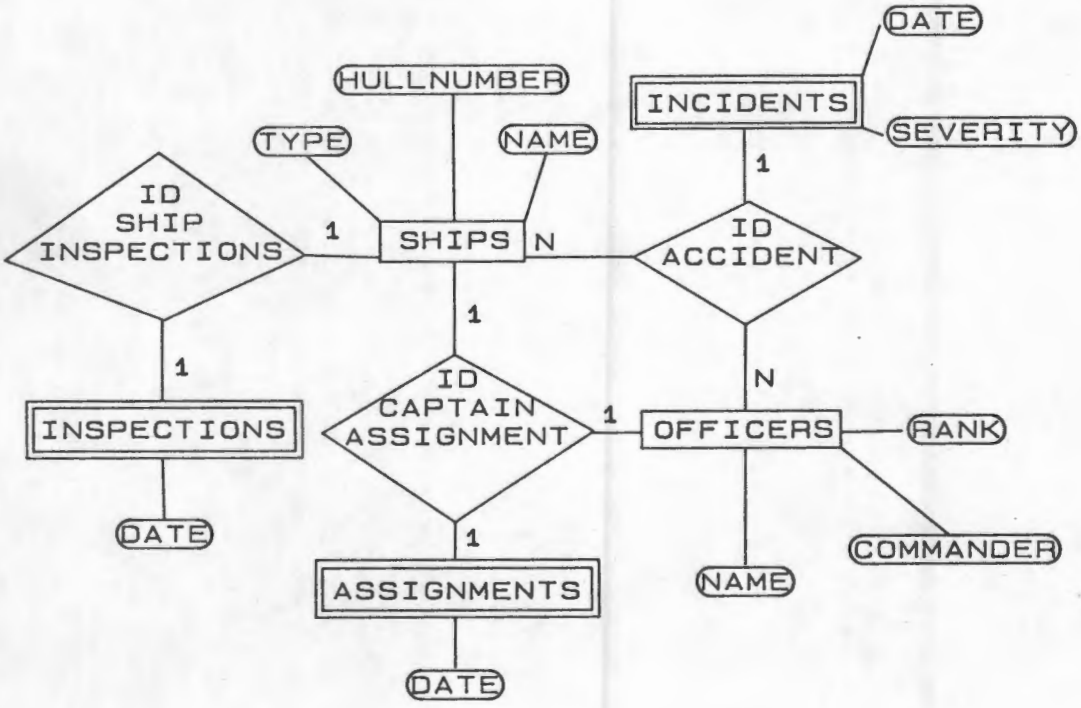


Figure 3.2 An Entity-Relationship Diagram for part of a Naval Database Description.

Among the models where real life is described using entities, properties, values and relationhips, Chen's is the most general; the only restriction is that relationships cannot themselves partake in relationships [Kershberg 1976]. It has however been criticised for not treating repeating attributes adequately and neglecting MVDs, and FDs between attributes [Shen 1978]. Whereas the concept of generalisation could be included, dynamics and integrity constraints are difficult to incorporate in a graphical model such as this. The concept of identifiers is also missing, but several authors such as Earnest [Earnest 1975] are of the opinion that keys are neither necessary nor desirable.

An entity-attribute-relationship model which breaks with the idea of graphical representation is the TAXIS model [Mylopoulos 1980], which describes a database-state as a 7-tuple (T,C,M,DP,FP,->,=>). T (Tokens) correspond to the values in the database such as "Smith" or 6. C is a collection of classes (i.e. entity types), for example OFFICERS and SHIPS; while M stands for the collection of "Metaclasses" whose instances are classes - this corresponds to generalised classes. DP is the set of definition properties associated with a class or metaclass. The DP [OFFICERS, weight, Person-Weight] specifies that (meta)class OFFICERS has property "weight" which maps it onto a Person-Weight value. FP is the collection of facts, such as ["Smith", weight, 54]. -> is termed "instance of" and gives the association between a class and the generic concept of which it is an occurrence. Finally, => is the generalisation relation, eg. OILTANKERS => SHIPS. TAXIS has been extended to include dynamics. These take the form of "exceptions" and "transactions", the latter allowing for the definition of variables, transaction preconditions, a transaction body and return values. An interesting aspect of this model is that

relationships are not directly represented as one of the seven criteria, but are included using definition properties. In this way they can be given role names and could include FD's and MVDs. (It is not clear from the literature whether or not dependencies are part of TAXIS; since no reference to this has been found it is presumed that they are not treated).

CSDL (a Conceptual Schema Definition Language) is actually a design tool for creating conceptual schemas [Roussopoulos 1979]. However, it incorporates a new type of entity-attribute-relationship model that is of interest in its own right. This involves "concepts" (objects) and "frames" (associations) that describe the characteristics and behaviour of an abstraction or situation. The semantic network model on which it is based consists of labelled, directed edges connecting nodes and subgraphs. Griffith [Griffith 1982] says "The existence of semantic networks is implied whenever information is conveyed in node-edge graphical form, where these nodes and edges are assigned meanings... Often they are called entity/relationship diagrams". In CSDL a label can be agent, object, source, destination, time, result, property of or argument of. An edge is labelled [all] , [n} , {n] or [n] to indicate how many concepts are involved in a mapping. Thus {n] means at most n, [n} at least n, etc.

Generalisation can be applied to both concepts and frames, using operators such as union, intersection and set difference to create new concepts and conjunction, disjunction and negation to define new frames. Constants can be used in definitions, such as BASICCOURSE a particular type of COURSE where LANGUAGE is 'BASIC'. Another important aspect of the CSDL model is that predicate calculus can be used instead of graphs, to define concepts and

frames. In this way a model designer can work in whichever manner he finds easiest, or use both methods. The non-mathematician would probably prefer the graphical version, but the problem there is that as the number of nodes grows, this becomes more awkward to use. A CSDL specification is shown in fig. 3.3.

```
concept MALESTU(X) derived
        frame MALESEX [(X/STU, male/SEX.VALUE) :
                        SEX (property of: X,
                             value: male)]
frame ASSIGNMENT [(X/LECTURER, [3} Y/COURSE,
                  [all] Z/DEPT):
                   ASSIGN (agent:Z, object:LECTURER,
                           destination: Y)]
frame ENROLLMENT = and (TEACH,RECEIVE.GRADE,<Z,Y>)
```

Figure 3:3 An example showing part of a CSDL model

This model can be seen to embody most of the features discussed earlier : naming, role names (which can be applied to entities, attributes, or relationships), integrity assertions, semantic relationships and generalisation. The latter is additionally powerful because it can be applied to associations as well. Facilities that are not included are dependencies and dynamics, both of which are however of doubtful value because of their complexity and impermanence, respectively. It has been said that "frames are not always a perfect fit to all instances of the pattern (assocation) involved. They cannot accomodate transient or unanticipated information and become obsolete if the actual patterns shift as time passes. Also, frames tend to be application oriented" [Griffith 1982]. Tsichritzis and Lochovsky claim that "Semantic networks are very rich data models in terms of concepts. This is a mixed blessing" because it complicates the

model [Tsichritzis 1982].

Several other entity-attribute-relationship models have been developed, such as that of Navathe and Schkolnik [Navathe 1978]. This is an extension of Chen's model where a variety of associations are defined, each with special update characteristics to incorporate integrity constraints. Other examples are SDM, EDSM [King 1981, King 1982] and DIAL [Hammer 1978]. SDM incorporates generalisation and integrity assertions, as well as other concepts such as grouping entities. EDSM (the Event Database Specification Model) is an extension of SDM to include dynamics. This takes the form of "events" having "working subtypes" (input entities with conditions on their attributes) and an "action box" defining the operations that form the event. DIAL is also an extension of SDM and is similar to a program with two sections: a database description (in modified SDM) and a collection of procedures that operate on the database. SDM will be discussed in detail in the next chapter; hence its description is deferred until then.

3.2.3. Binary relationship models.

A binary relationship model is built on the concept of linking together two objects, which could be entities, attributes, relationships or values. Associations between more than two objects are split into several binary associations. Such a model is graphically represented with nodes for entities and attributes, connected by links representing relationships. A surprisingly large number of these models have recently been developed, probably because of the appealing simplicity of binary

relationships. Among the most well-known are those of Abrial, Grabowski and Eigner, Chang and Cheng, and Su and Lo [Abrial 1974, Grabowski 1979, Chang 1978, Su 1979].

In Abrial's model each connection is labelled by "access functions" describing the asociation and its inverse. The operator INV is used if the designer cannot find a name for the inverse access function, thus somewhat reducing the difficulty of continually requiring names for all functions. An access function gives the minimum and maximum number of target elements e.g. SEX = (1,1), PERSONSOFSEX = (0,infinity), PARENTSLIVING = (0,2) etc. Thus special relationships such as HAS-PARENTS can be explicitly stated to map to exactly two target entities, a facility not available in many other models. Although this model does not handle generalisation, FDs or MVDs, it does have the advantage that axioms and theorems have been developed to provide rules for "well-defined" models. It does support semantic relations. Most of the constraints in the model are specified via operations which are similar to programming language constructs, using FOR loops, etc. This model has been criticised for not being sufficiently user-oriented [Tsichritzis 1982].

Grabowski and Eigner [Grabowski 1979] also base their model on binary relationships, which can be entity-entity, entity-association or association-association. The second of these is to represent the meaning of an object within a relationship, the last to show access preference, eg. access relationship JOB before relationship JOBHISTORY. They are labelled 1:1,1:n,n:n or n:1 and are also given a "role" (either dependent - the target cannot exist unless linked to some source - or indpendent) and a set function. The latter governs the

compulsory or optional nature of the association and its inverse. This can be stipulated as SOME-ALL, SOME-SOME, ALL-SOME or ALL-ALL.

Every entity must have at least one identifier. An attribute, which is given a definition range, can refer to an entity or association. The model includes four types of integrity constraint and also incorporates the concept of generalisation. However, relationships are not named and neither are their inverses. Derived information is not catered for, and the model does not allow the user much freedom of interpretation.

Database skeletons [Chang 1978] are an extension of the directed graph model of Wang and Wedekind [Wang 1975] where the links reperesent binary relationships. This model consists of two basic elements : attributes and relationships. These elements are described by contents rules (giving type, format and characteristic values) and semantic or functional relations, respectively. Associations can be attribute-attribute or attribute-relationship or relationship-relationship. Functional relations include key compatibility, full and functional dependencies, similarity (i.e. drawn from the same domain), subfile relation(i.e. one relationship can be decomposed into a collection of other relationships), set-type relation (essentially 1;many), hierarchical-type relation (basically many:many) and "leads to" (Ri leads to Rj if a key of Ri belongs to Rj). Semantic relations are analagous to "roles" in the entity-relationship model : they give a meaningful name to a relationship. Propositional calculus is used to specify functional relations and contents rules. An example is given in fig.3.4. Except for dynamics and generalisations, this model

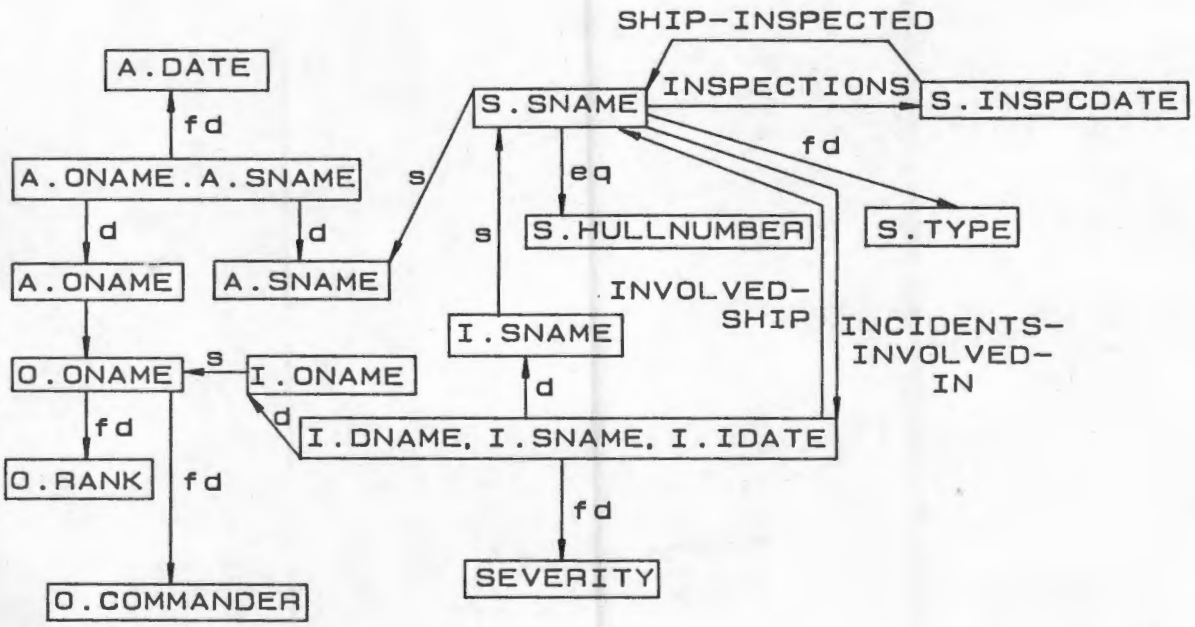


Figure 3.4 The database skeleton for part of a naval database description.

encompasses all the features described earlier. It is however not as easy to design database skeletons as it is to create other binary relationship models, chiefly because some of the functional relations are too complicated. The Semantic Association Model [Su 1979, Su 1981] involves networks of atomic concepts (character strings) and non-atomic concepts (associations between atomic concepts which can represent objects, relationships, types, subtypes, etc). This model also includes the specification of integrity assertions for each type of relationship. Several types of "semantic association" are supported, such as 'set relationship' which corresponds to generalisation, 'cause effect' to relate say VALID-CLAIM and INSURANCE-PAYMENT, 'action purpose' to associate say SALES-VIST and ADDING-CLIENT, etc. This model does not support FDs or MVDs, but is semantically rich in both its statics and dynamics.

The CAZ-graph introduced by Zaniolo and Melkanoff [Zaniolo 1982] is also a binary relationship model. An example of a CAZ-graph is given in fig.3.5. CAZ-graphs comprise nodes representing attributes and edges representing relations. Relations of degree N, N > 2, are represented by several arcs, each connecting two attributes. Each edge is labelled by a number indicating the relation in which the attributes connected by that edge appear. Edges have one arrow to indicate 1:many, two arrows to indicate 1:1, and no arrows to indicate many:many relationships, respectively. To determine whether a relationship from an attribute A1 to A2 is 'to 1' or 'to many' in an N-ary relationship (N > 2), the remaining attributes are held constant. As only attributes are used in CAZ-graphs, Zaniolo and Melkanoff suggest that after translating a relation scheme into such a graph, attributes should be grouped together as entities where necessary

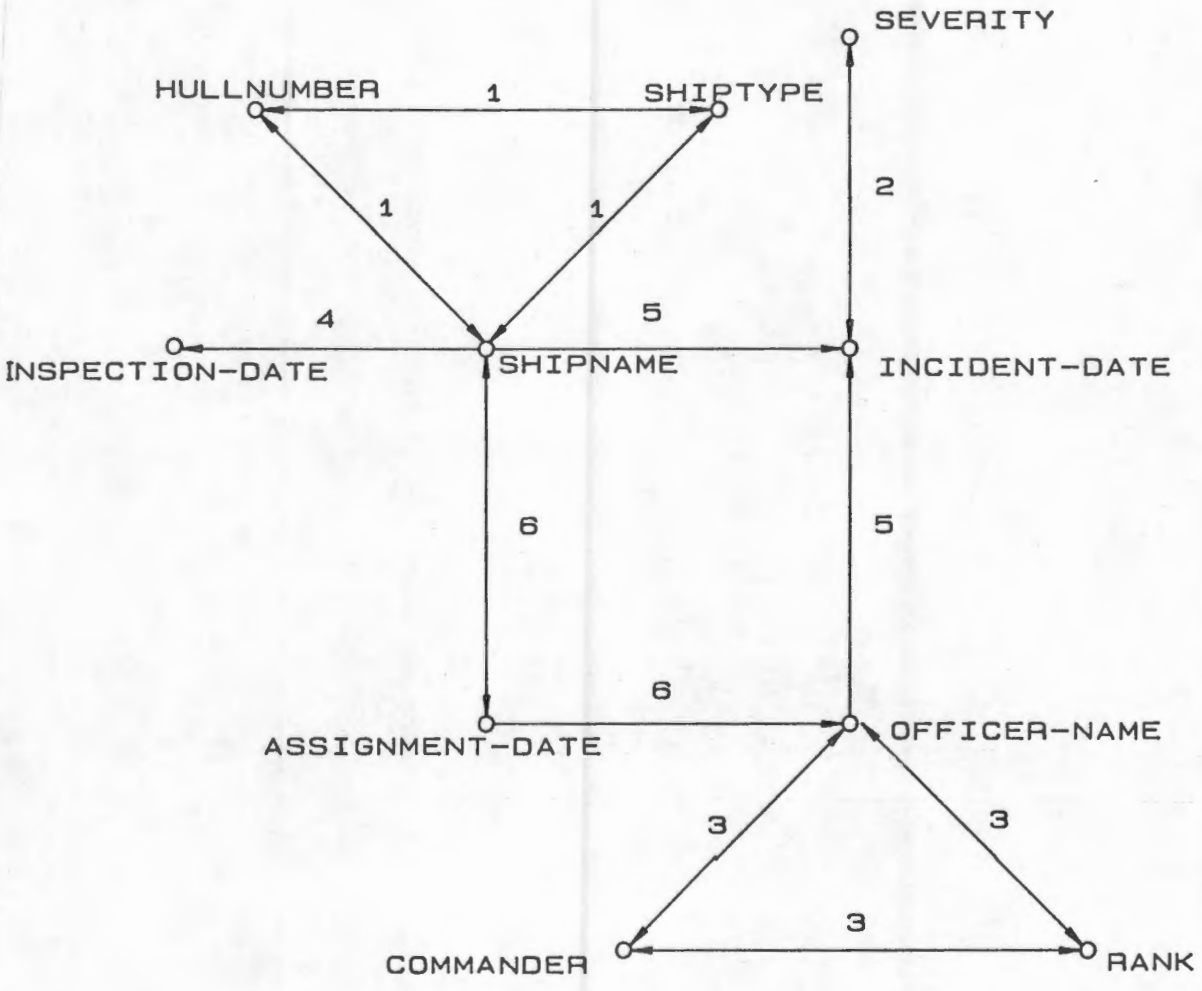


Figure 3.5 A CAZ-graph for a section of a naval database.

and the graph adjusted accordingly. The benefit derived from CAZ-graphs is that they make binary relationship diagrams amenable to rigorous definition, since a CAZ-graph is the representation of a relation scheme. However, this model suffers from the same problems as other binary relationship models : limited semantics and no derived information is represented.

3.2.4. Aggregation.

The idea of building a data model recursively was introduced by Smith and Smith [Smith 1977a and 1977b]. Such a model is constructed by recursive definition of aggregates. THat is, relationships between objects are considered as higher-level, named objects. An aggregate must be given a unique natural language noun as a name, and represents any object - entity, attribute, association or some combination thereof. Since an aggregate is a named relationship between other aggregates, the model is structured as a hierarchy of aggregates (see fig. 3.6). Each has a key, and embedded aggregates which correspond to a useful abstraction are separated out.

One advantage of this model is that removal of embedded aggregates can be used to ensure aggregates are in 3rd normal form. The motivation for using aggregates is that they enable a designer to deal with a minimum of detail, as an aggregate name can be thought of without worrying about the underlying relationship. Although this model neglects dependencies and includes a minimum of integrity assertions, it does have a set of rules for "well defined" aggregate hierarchies against which a model can be verified. The model has been extended to include transaction

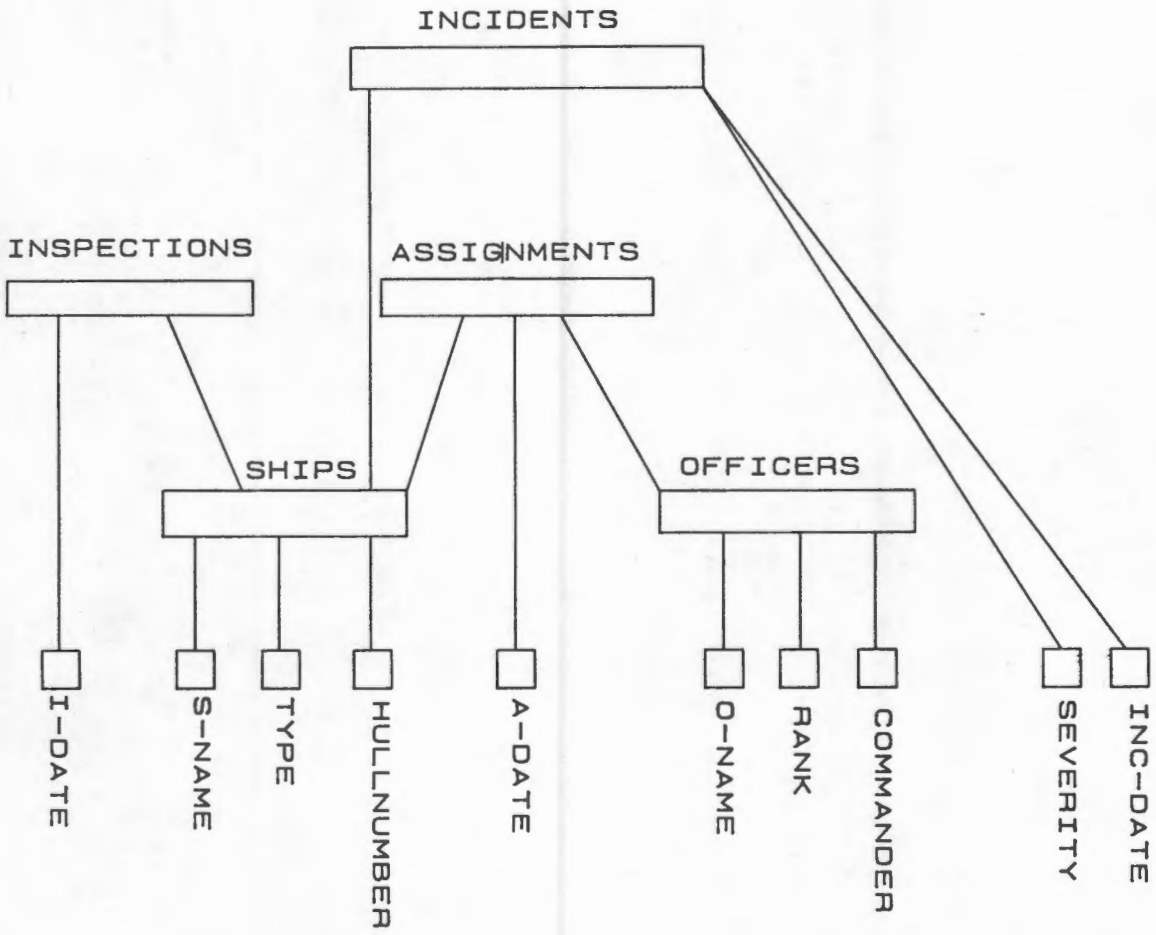


Figure 3.6 Aggregation model for part of a naval database.

specification by means of functions and procedures. These are defined using control structures such as IF... THEN... ELSE..., and a predicate language. A few other models fall into this category such as that developed by Bracchi [Bracchi 1976]. This consists of hierarchies of concepts, a concept defined as the "smallest unit which is amenable to processing".

3.2.5. Extended Relational Models.

Some models are extended versions of the relational model - such as Codd's RM/T [Codd 1979] and the Structural Model of Wiederhold and El-Masri [Wiederhold 1979]. These use a relation to represent an entity and its attributes, or an association; and then incoporate some semantics into these relations. Codd for example uses an "entity relation", having only one column, for each entity - this specifies its internal unique identifier. He also treats single and multi-valued attributes, application events and entity-groups. Wiederhold and El-Masri have special relations for "existence dependencies" (eg. an INSURANCE-CLAIM cannot exist if its POLICY has been deleted) and "lexicons" to represent one-to-one correspondence between identifiers (eg. EMPLOYEE-NUMBER and IDENTITY-NUMBER) and includes integrity rules.

3.2.6. Functional Data Models.

Several models have also been developed to represent not only the statics of an application, but also the "dynamics". Models which base their static properties on the dynamics of the application are termed functional database models. This type of model was introduced by Sibley and Kerschberg [Sibley 1977] in 1977. Here,

attributes are viewed as mappings from an object to other objects. Function constraints such as single- or set-valued, total or partial, one:one or one:many etc can be specified. Models in this category are DAPLEX, FQL and FDM. DAPLEX [Shipman 1981] uses zero-argument functions to define types, single- or multi-argument functions to define associations. It includes a data manipulation language for manipulating functions using predicates and allows the data definition itself to be queried. Its most appealing feature is its concise syntax which avoids awkwardness. A function returns one or more entities, not their keys; it also returns their role and their 'order'. The latter is used in DAPLEX DML constructs such as "FOR EACH ... IN ORDER", "PRECEEDING" and "FOLLOWING". DAPLEX provides for separate user views, which can be converted to the original view by means of derived functions. Derived functions are also used for other purposes, to define unions, intersections, differences, cartesian products, recursion, averages and boolean or arithmetic expressions. Generalisaion is neatly included in the model, as are constraints. The latter are special functions for which a transaction aborts if the function value is 'false'. The conciseness and clarity of the notation can be seen in the example given in fig. 3.7.

```
DECLARE Person() =>> ENTITY
DECLARE Name(Person) => STRING
DECLARE Student() =>> Person
DECLARE Dept(Student) => Department
DECLARE Course(Student) =>> Course
DEFINE Students(Course) =>> INVERSE OF Course(Student)
```

Figure 3.7 Part of the Daplex model of a University Database.

A similar approach is adopted in FQL [Buneman 1979], which is a query language based on a functional data model. The notation is based on the functional programming notation advocated by Backus [Backus 1978]. This allows users to invert or form compositions of mappings, as well as to derive mappings that return instances of objects. As an example of the latter, one could construct a function that takes INCIDENT and derives the SHIP-INVOLVED and INCIDENT-DATE. Another functional model FDM [Housel 1975] provides data manipulation facilities which treat mappings as access paths and incorporate generalisations as well as other functions.

3.2.7. Infological Data Models.

The infological data model has been extensively studied [Kahn 1976, Kahn 1978, Teichroew 1977]. Langefors' model [Langefors 1963,1969,1974,1975,1977,1980] will be considered here. The infological model attempts to make model specification as natural as possible - it tries to embody the way that people think about facts. The real world is described by means of 'elementary facts', represented by a triple (o,r,t) called an 'elementary constellation'. The o is a tuple of objects, r is a relationship or property, and t represents time (which is emphasised throughout this model). Objects can be constellations, hence allowing facts to be structured in many different ways. It is thus easy to associate a property with a relationship, without needing to treat that relationship as an entity. "The resulting structure can be visualised as a very complicated graph ... where the nodes are objects, properties, relationships and time" [Tsichritzis 1982]. Rules by which properties and relationships can be combined to

form new properties and relationships exist, and hence both fundamental and derived properties or relationships can be used.

This model is the epitome of loosely typed models, as there is no definition of types of objects, properties or relationships to which all facts have to conform. A clear distinction is made between an object and a reference to that object. An elementary message is a triple (x,y,z) where x is a tuple of unique object references, y is a unique property/relationship r and z is a time reference. This is different from a constellation, as it deals with references to objects, while constellations deal with objects themselves. To physically represent an infological model, an elementary message is represented by an 'elementary record', several of which can be combined to form 'compound records'. Sundgren also include virtual elementary records in his infologial model [Sundgren 1974]. These are not physically stored but are deducible from other stored data.

Dynamics constructs exist in the form of 'elementary processes', which create new elementary messages from given messages. These comprise a set of (input) preconditions and a list of actions. Syntactic constraints in an infological model compensate for its loosely typed nature. Facts are divided into three categories: false, true and meaningful. 'Meaningful' ones can be included in the model even although they may be untrue; a message is thus more of an 'opinion'. Elementary messages are always interpreted within the context of some constellation. Hence a fact may be true in one context and false in another. 'Filters' enforce constraints on how a message can be interpreted, and can thus protect users from seeing untrue or inconsistent information. They also serve to enforce privacy and security. The

implementation of filters is not defined. This is because in infological models, the view is held that the individual implementors should make these decisions.

It must be remembered that other models have also been suggested. These cannot readily be placed in any of the categories discussed and differ greatly from the better-known models just described. For example, Heyderhoff [Heyderhoff 1978] advocates a Petri-Net model. The model suggested by Jardine and Steele [Jardine 1980] is based on Interpreted Predicate Logic. This has two basic constructs : entities and predicates. An attribute is specified as a predicate on the appropriate entity; a relationship as a predicate on the participating entities; etc.

3.3. Comparing Data Models.

An attempt to compare these models is similar to arguing the relative merits of programming languages. "It is very difficult to argue persuasively that one data model is best uniformily. Each data model has advantages, depending on who is doing the schema design and the realm in which one is working" [Tsichritzis 1982]. From all the models presented, a short list was constructed from which to select a starting point for the ADD system. This comprised the models of Chen, Smith and Smith, Roussopoulos, Chang and Cheng, Abrial, Hammer and McLeod, and Grabowski and Eigner. A comparison of these models is deferred to the next chapter, when the choice of a model for the ADD system is discussed.

[Abrial 1974] Abrial, J.R. "Data Semantics", in Database Management (eds. Klimbie, J. and Koffeman, K.) , North Holland, 1974.

[Appleton 1980] Appleton, D.S. "Implementing Data Management", AFIPS NCC Conf. Proc., 1980, pp. 307-316.

[Backus 1978] Backus, J. "Can programming be liberated from the von Neumann style? A functional style and its algorithm of programs", Comm. of the ACM Vol. 21, No. 8, August, 1978, pp. 613-641.

[Bourne 1979] Bourne, T.J. "The Data Dictionary System in Analysis and Design", ICL Technical Journal, Vol. 1, No. 3, November 1979, pp. 292-298.

[Bracchi 1976] Bracchi, R.F., Paolini, P. and Pelagatti, F. "Binary Logical Associations in Data Modelling", in Modelling in Database Management Systems, (Ed. Nijssen, G.M.), North-Holland, 1976, pp.125-148.

[Buchman 1979] Buchman, A.P. and Dale, A.G. "Evaluation Criteria for Logical Database Design Methodologies", Computer-Aided Design, Vol. 11, No. 3, May 1979, pp. 121-126.

[Buneman 1979] Buneman, P. and Frankel, R. "FQL: A Functional Query Language", Proc. of ACM SIGMOD Int. Conf. on the Management of Data, Boston MA, 1979.

[Chang 1978] Chang, S.K. and Cheng, W.H. "Database SKeleton and its Applications to Logical Database Synthesis", IEEE Trans. on Software Engineering, Vol. SE-4, January 1978, pp. 18-30.

[Chang 1978b] Chang, S.K. and Ke, J.S. "Database Skeleton and its Application to Fuzzy Query Translation", IEEE Trans. on Software Engineering, Vol. SE-4, January 1978, pp. 31-43.

[Chen 1976] Chen, P.P.S. "The Entity-Relationship Model: Toward a Unified View of Data", ACM Transactions on Database Systems, Vol. 1, No. 1, March, 1976, pp. 9 - 36.

[Chen 1977] Chen, P.P.S. "The Entity-Relationship Model: A Basis for Enterprise View of Data", Proc. AFIPS NCC, Vol. 46, 1977, pp. 77-84.

[Chen 1980] Chen, P.P.S.(editor) "Entity-Relationship Approach to Systems Analysis and Design", North-Holland, Amsterdam, 1980.

[Codd 1979] Codd, E.F. "Extending the Database Relational Model", ACM Trans. on Database Systems, Vol. 4, No. 4, December 1979, pp. 397 - 434.

[Earnest 1975] Earnest, C.P. "Selection and Higher Level Structures in Database Systems", in Database Description, (eds. Nijssen, G.M. and Douque, B.C.M.), North-Holland, Amsterdam, 1975, pp. 215-237.

[Grabowski 1979] Grabowski, H. and Eigner, M. "Semantic Datamodel Requirements and Realization with a Relational Datastructure", Computer-Aided Design, Vol. 11, No. 3, May 1979, pp. 158-167.

[Griffith 1982] Griffith, R.L. "Three Principles of Representation for Semantic Networks", ACM Trans. on Database Systems, Vol. 7, No. 3, September 1982, pp. 417-442.

[Hammer 1978] Hammer, M. and Berkowitz, B. "DIAL: A Programming Language for Data-Intensive Applications", Proc. of ACM SIGMOD, Austin Texas, May 1978.

[Heyderhoff 1978] Heyderhoff,P. "Comment on Database Design: Towards an Integrated Methodology by Bo Sundgren", in Database Management (eds. Weber,H. and Wasserman, A.I.), North-Holland, Proc. 4th Conf. on Very Large Databases, 1978.

[Housel 1975] Housel, B.C. et al, "DEFINE - A Nonprocedureal Language for Defining Information Easily", Proc. ACM Pacific 1975, pp. 62-70.

[Jardine 1980] Jardin, D.A. and Steele, T.B. Jr., "ISO Report on Concepts for Conceptual Schemas", Proc. of the 6th Int. Conf. on Very Large Databases, Montreal, 1980, pp. 321-325.

[Kahn 1976] Kahn, B.K. "A Method for Describing Information Required by the Database Design Process", Proc. ACM SIGMOD, 1976, pp. 53-64.

[Kahn 1978] Kahn, B.K. "A Structured Logical Database Design Methodology", Proc. NYU Symposium on Database Design, 1978, pp. 15-24.

[Kershberg 1976] Kershberg, L., Klug, A. and Tsichritzis, D.C. "A Taxonomy of Data Models", in Systems for Large DataBases, (eds. Lockeman, P.C. and Neuhold, E.J.), North-Holland, 1976.

[King 1981] King, R. "The Event Database Specification Model", Ph.D. Thesis, Computer Science Department, University of Southern California, Los Angeles CA, 1981.

[King 1982] King, R. and McLeod, D. "An Approach to Database Design and Evolution:, in Database Modelling, (eds. Brodie, M., Mylooulos,J. and SChmidt,J.), 1982.

[Langefors 1975] Langefors, B. "Control Structure and Formalized Information Analysis in an Organization", in Information Systems and Organizational Structure, (eds. Grochla, E. and Szyperski, N.), Walter de Gryter, New York NY, 1975, pp. 311-322.

[Langefors 1977] Langefors, B. "Infological Models and Information User Views", Inf. Syst. Vol. 5, pp. 17-32.

[Langefors 1977] Langefors, B. "Information Systems Theory", Inf. Syst. Vol. 2, 1977, pp. 207-219.

[Langefors 1974] Langefors, B. "Information Systems", Proc. IFIP Congress 1974, North-Holland, Amsterdam, pp.937-945.

[Langefors 1969] Langefors, B. "Management Information Systems Design", IAG Quart. Vol. 2, No. 4, 1969, pp. 7-17.

[Langefors 1963] Langefors, B. "Some Approaches to the Theory of Information Systems", BIT Vol. 3, 1963, pp. 1-79.

[Ledgard 1977] Ledgard, H.F. and Taylor, R.W. "Two Views of Data Abstraction", Comm. of the ACM, Vol. 20, No. 6, 1977, pp. 382-384.

[McLeod 1980] McLeod, D. "On Conceptual Database Modelling", Proc. of Workshop on Data Abstraction, Databases and Conceptual Modelling, Pingree Park Co, June 1980.

[Mylopoulos 1980] Mylopoulos, J. and Wong, H.K.T. "Some Features of the TAXIS Data Model", Proc. of the 6th Int. Conf. on VLDB, Montreal, October 1980, pp. 399-410.

[Navathe 1978] Navathe and Schkolnich "View Representation in Logical Database Design", Proc. of ACM SIGMOD Int. Conf. on the Management of Data, Austin TX, June 1978.

[Roussopoulos 1979] Roussopoulos, N. "Tools for designing Conceptual Schemata of Databases", Computer-Aided Design, Vol. 11, No. 3, May 1979, pp. 119-120.

[Shen 1978] Shen, S.N.T. "A Semantic Approach in Designing Relational Databases", ACM 1978 Annual Conf. Proc., pp. 596-601.

[Shipman 1981] Shipman, D. "The Functional Data Model and the Data Language DAPLEX", ACM Trans. on Database Systems, Vol. 6, No. 1, March 1981, pp. 140 - 173.

[Sibley 1978] Sibley, E.E. and Kershberg, L. "Data Architecture and Data Model Considerations", Proc. AFIPS NCC, Dallas TX, June 1977, pp. 85-96.

[Smith 1977a] Smith, J. M. and Smith, D. C. P. "Database Abstractions : Aggregation", Comm. of the ACM, Vol. 20, No. 6, pp. 405 - 413, June 1977.

[Smith 1977b] Smith, J. M. and Smith, D. C. P. "Database Abstractions : Aggregation and Generalization", ACM Trans. on Database System Vol. 2, No. 2, June 1977, pp. 105-133.

Steele, ?. "?", in Database Management (eds. Weber,H. and Wasserman, A.I.), North-Holland, Proc. 4th Conf. on Very Large Databases, 1978.

[Su 1981] Su, S.U., Lam, H. and Lo D. H. "Tranformation of Data Traverals and Operations in Application Programs to Account for Semantic Changes of Databases", ACM Trans. on Database Systems Vol. 6, No. 2, June 1981.

[Su 1979] Su, S.Y. and Lo, D. H. "A Semantic Association Model for Conceptual Database Design", Proc. of Int. Conf. on the Entity-Relationship
Approach to Systems Analysis and Design, Los Angeles CA, December, 1979.

[Sundgren 1974] Sundgren, B. "Conceptual Foundation of the Infological Approach to Data Bases", in Data Base Management (eds. Klimbie, J.W. and Koffeman,
K.L.), North-Holland, Amsterdam, 1974, pp. 61-96.

[Sundgren 1978] Sundgren, B. "Data Base Design in Theory and Practice: Towards an Integrated Methodology", Proc. 4th Int. Conf. on Very Large Databases, 1978, pp. 3-16.

[Teichroew 1977] Teichroew, D. and Hershey, E.A. "PSL/PSA: A Computer-Aided Technique for Structured Documentation and Analysis of Information Processing Systems", IEEE Trans. on Software Engineering, Vol. SE-3, 1977, pp. 41-48.

[Trauboth 1978] Trauboth, H. "Database Design in Software Systems for Process Monitoring and Control", in Database Management, (eds. Weber, H. and Wasserman, A.I.), North-Holland, Proc. of the 4th Int. Conf. on Very Large Databases, 1978.

[Tsichritzis 1982] Tsichritzis, D.C. and Lochovsky, H. "Data Models", Prentice Hall, Englewood Cliffs NJ, 1982.

[Wang 1975] Wang, C. and Wedekind, H. "Segment Synthesis in Logical Data Base Design", IBM Jnl. of Res. and Development Vol. 19, January 1975, pp. 71-77.

[Wasserman 1978] Wasserman, A.I. "A Software Engineering Approach to Database Management", in Database Management (eds. Weber, H. and Wasserman, A.I.), North-Holland, Proc. 4th Int. Conf. on Very Large Databases, 1978.

[Zaniolo 1976] Zaniolo, C. "Analysis and Design of Relational Schemata for Database Systems", Report UCLA-ENG-7669, Computer Science Department, University of California, Los Angeles, July 1976.

[Wiederhold 1979] Wiederhold, G. and El-Masri, R. "Structural Model for Database Design",
Proc. of Int. Conf. on the Entity-Relationship Approach to Systems
Analysis and Design, Los Angeles CA, December 1979.

[Zaniolo 1982] Zaniolo, C. "A New Normal Form for the Design of Relational Database Schemata", ACM Trans. on Database Systems, Vol. 7, No. 3, September 1982, pp. 489-499.

4. CHAPTER 4. SDM - THE SEMANTIC DATABASE MODEL.

This chapter presents the modelling principles on which SDM is based, and then gives a detailed description of this data model. The reasons for choosing SDM for the ADD system are also explained.

4.1. General Principles.

SDM was developed by Michael Hammer and Dennis McLeod and first appeared in the literature in 1978. It has been described as "a high-level semantics-based database description and structuring formalism (database model) for databases" [Hammer 1978]. The principles on which it is based are relativism and logical redundancy. SDM supports a relativist view of the meaning of data, allowing multiple ways of looking at the same information. Most data models do not support relativism. For example, they should allow a relationship to be represented as an attribute of any, or all, the objects involved; or as an entity in its own right. SDM enables either or both of these representations to be used, and introduces a special construct called matching to indicate that they are describing the same association. Redundancy within the model enables users to describe their organisation in the manner most natural to them, without being restricted to only one method. The freedom afforded the designer will become evident as the model is described in detail.

Logical redundancy enables items that are algorithmically derivable from other items to be included in the database. This facilitates the development of applications since values that

would otherwise have to be repeatedly computed are actually stored on the database. It also makes it possible for users to treat derived data properties as if they were primitive, thereby enhancing the naturalness of the model. Integration is necessary to control this redundancy. An integrated description precisely defines the relationships between alternate views of the same data, and between an item and those items from which it is derived.

4.2. A Description of SDM.

The SDM model comprises two basic constructs: entities, called 'classes', and attributes. Some classes can be defined as subclasses or groupings of another class. The former embodies the concept of generalisation and the latter allows an entity type to be defined as a collection of like entities. (Example: CONVOYS are groupings of SHIPS). Each of these four constructs will be described and then the various ways of representing relationships in SDM will be presented. A complete SDM specification for a naval database is given in Appendix D.

4.2.1. Classes.

The entities in the application environment are organised into 'classes', the SDM terminology for an entity type. Individual occurrences of an entity type are called 'members' of that class. Classes represent objects, events or relationships. In an SDM specification each class can be given a description (serving as documentation to define the meaning of that class), a list of

member and class attributes, and an interclass connection. A member attribute is a property that members of this class have, such as NAME, SEX, AGE. A class attribute is one that applies to the class as a whole. Class attribute AVERAGE-AGE of OFFICERS would store the average age value over all members of OFFICERS.

A class is said to be 'base' if it is "defined independently of all other classes in the database; it can be thought of as modelling a primitive entity" [Hammer 1978]. Non-base classes are subclasses or grouping classes. Their relation to other class(es) in the database is explicitly defined by an interclass connection, as explained in the next two sections. Base classes must have the following additional information: a list of attributes (or attribute combinations) which identify a member of that class, and a specification of whether duplicates are allowed. Value types, such as COLOURS or PERSON-NAMES, are defined by means of special classes termed name classes. These are all defined as subclasses of a pre-defined name class, "STRINGS".

4.2.2. Subclasses.

Every subclass must be defined using one of four kinds of subclass connection. The class of which it is a subclass is called its superclass. For example subclass RURITANIANSHIPS of superclass SHIPS can be defined as 'where COUNTRY-OF-REGISTRY = "RURITANIA"', COUNTRY-OF-REGISTRY being an attribute of SHIPS. This is an example of the most common subclass connection, the expession-defined subclass. Here conditions on the values of attributes in the superclass determine whether a member belongs to the subclass.

A subclass can be the union, intersection or difference of two other classes (as long as all three are subtypes of the same base class). For example, subclass SHIPS-TO-BE-MONITORED of class SHIPS can be specified as 'where is in BANNEDSHIPS or is in OILTANKERS-REQUIRING-INSPECTION'. What is known in SDM terminololgy as class difference actually involves taking complements, as in subclass SAFESHIPs being defined 'where is not in BANNEDSHIPS'.

Another form of subclass connection is an 'existence subclass' where members are assigned to the subclass according to their association with other entities in the database. As an example, subclass DANGEROUS-CAPTAIN can be defined 'where is a value of INVOLVED-CAPTAIN of INCIDENTS'. In this case INCIDENTS would be a class with attribute INVOLVED-CAPTAIN, representing some relationship between a captain and an incident.

The final type of subclass connection is termed user-controllable and caters for special subclasses where the user is free to specify which members of the superclass belong to the subclass - i.e. this is independent of their attributes and relationships to the rest of the data. BANNEDSHIPS would be defined in this way, giving the user the authority to decide which SHIPS to ban, independent of any information on these SHIPS in the database. It is essential to grant the user the ability to specify subclasses as user-controllable for circumstances such as this.

When defining name classes, which are either subclasses of STRINGS or of other name classes, any of these subclass connections can be used except for "existence subclasses". This is clearly not applicable, as name classes represent value types which cannot participate in relationships.

4.2.3. Grouping Classes.

There are three ways of specifying how the members of a class can be grouped together to form a new type of class. The first is called an expression-defined grouping and groups members according to their values for designated attribute(s). As an example, the class CARGOTYPE-GROUPS can be a grouping of SHIPS 'on common value of CARGOTYPES'. The members of CARGOTYPE-GROUPS are not SHIPS, they are groups of SHIPS. If there are n different CARGOTYPES, there would be n CARGOTYPE-GROUPS members, each comprising a set of SHIPS. The attributes used in an expression-defined grouping may be single- or multi-valued. Hence the aforementioned description of CARGOTYPE-GROUPS would also be valid if a ship had more than one cargotype: a ship would simply participate in more than one "CARGOTYPE-GROUPS" group.

When no attribute(s) would appropriately define a group, an 'enumerated grouping class' is used, which lists other classes already defined. To be meaningful, these must all be defined on the same (eventual) base class. For example class TYPES-OF-HAZARDOUS-SHIPS can be described as 'grouping of SHIPS consisting of classes SHIPS-TO-BE-MONITORED, BANNED-OILTANKERS, and OILTANKERS-REQUIRING-INSPECTION.' The remaining grouping class connection is the user-controllable grouping class, which enables members to be placed into groups at the discretion of the user. This caters for the situation where there is no rule or condition for deciding how to form groups. Class CONVOY is an example of a user-controllable grouping of SHIPS.

4.2.4. Attributes.

Several properties of attributes can be included in an SDM specification, thus providing a semantically rich description of the data in the application environment. Each attribute can have a 'description' associated with it to explain its meaning. The following five properties may be specified for an attribute:

(1) multi-valued: if it can assume more than one value. SHIPSENGINES will be a multi-valued attribute if a ship has several SHIPSENGINES.

(2) mandatory: if it may not be null. For example, OFFICER-NAME should be specified as mandatory since this item should never be blank.

(3) not changeable. This means once stored, its value cannot be altered. This is meaningful for item BIRTHDATE, which is clearly a permanent characteristic of any person.

(4) exhaustive of valueclass: if attribute SHIPSENGINES of class SHIPS exhausts valueclass then every ENGINE in the database must be the SHIPSENGINE of some SHIP.

(5) nonoverlapping. If the multi-valued attribute INSPECTIONS-OF-THIS-TANKER of class OILTANKER is nonoverlapping, the same INSPECTION cannot involve more than one tanker.

Every attribute must be designated as either a member attribute or a class attribute. It must also be given a 'valueclass', which is either an entity type or a name class. This tells from which domain the value of the attribute is drawn. Examples: attribute CAPTAIN of class SHIPS has as its valueclass the class OFFICERS, hence a CAPTAIN is drawn from the OFFICERS in the database. OFFICER-NAME has valueclass PERSON-NAME, indicating that it takes

on as value a character string of the type specified by name class PERSON-NAME. Throughout this thesis, the term 'standard attribute' refers to an attribute with a name class as its valueclass; others will be called 'non-standard'. Thus standard attributes are printable values (strings or numbers).

The final property of attributes enables the relationship of an attribute to other items in the database to be defined. This will be discussed in the next section.

4.2.5. Attribute Derivations.

As explained, the purpose of an attribute derivation is to define a relationship that exists between a 'derived' attribute X and other attributes. Usually X will be defined directly in terms of some other attribute in the same class. However it is possible for it to be related to attributes of other classes. To specify these 'indirect' relationships a special construct called a mapping is used. This will be briefly described and then each type of attribute derivation will be discussed in turn.

A mapping is a simple but extremely powerful tool in SDM. It is a concatenation of attribute names and allows attributes of attributes to be referenced. As an example, 'CAPTAIN.SENIORITY' refers to the SENIORITY of a ship's CAPTAIN. This mapping is possible because SHIPS has attribute CAPTAIN whose valueclass is OFFICERS, and OFFICERS has attribute SENIORITY. In other words the SHIP and SENIORITY values are indirectly related through the ship's CAPTAIN, and this is reflected in the mapping. Multi-valued attributes can also be used in a mapping. Thus 'CAPTAIN.SUPERIORS.DATE-COMMISSIONED' denotes all DATEs on which

the SUPERIORS of the ship's CAPTAIN were appointed. In the remainder of this chapter 'mapping' can be taken to mean an attribute or mapping, since an attribute is a special type of mapping.

4.2.5.1. Member Attribute Derivations.

A derivation is used to define precisely how an attribute is related to other information. The nine derivation primitives will be demonstrated by means of examples.

(1) ordering: attribute SENIORITY of OFFICERS can be defined as 'order by increasing DATE-COMMISSIONED'. The OFFICER with the 'lowest' DATE-COMMISSIONED will have a SENIORITY value of one, etc. Attribute ORDER-FOR-TANKER of INSPECTIONs can be specified as 'order by decreasing INSPECTION-DATE within TANKER-INSPECTED.' In this case, class INSPECTIONs has (at least) three attributes: ORDER-FOR-TANKER, INSPECTION-DATE and TANKER-INSPECTED. The ORDER-FOR-TANKER attribute will have a value of one if that particular INSPECTION has the 'greatest' INSPECTION-DATE of all the INSPECTIONs for that tanker (TANKER-INSPECTED). An ordering can have several mappings specified for determining the order, and can also be calculated 'within' several mappings.

(2) existence: such an attribute can only have one of two possible values, YES or NO. It is used to indicate whether an object belongs to a particular subclass. Suppose BANNED-OILTANKERS is a subclass of OILTANKERS. The OILTANKERS attribute IS-TANKER-BANNED with derivation 'if in BANNED-OILTANKERS' has value YES if the OILTANKER is a member of that subclass, else NO.

(3) recursion: if attribute COMMANDER of class OFFICERS has

OFFICERS as its valueclass, then attribute SUPERIORS of OFFICERS could be specified as 'all levels of values of COMMANDER.' This would give his COMMANDER, and his COMMANDER's COMMANDER, etc. In SDM 'up to N levels' can be stated, N any positive integer. In this case the recursion would only be performed N times when determining SUPERIORS.

(4) same: This simply indicates that the attribute is equivalent to some mapping i.e. Is identical to some other information in the database. An example is DATE-LAST-EXAMINED which is the 'same as LASTINSPECTION.INSPECTION-DATE'. Here INSPECTION, the valueclass of LASTINSPECTION, has an attribute called INSPECTION-DATE; DATE-LAST-EXAMINED and LASTINSPECTION are attributes of OILTANKER. This indicates that an OILTANKER's DATE-LAST-EXAMINED must equal the date associated with its LASTINSPECTION.

(5) subvalue: This defines a subset of a multi-valued attribute. (Recall that a multi-valued attribute is one which assumes not one but several values: it is analagous to an item with an OCCURS M TIMES clause in COBOL, M > 1). LAST-TWO-INSPECTIONs can be defined as 'subvalue of INSPECTIONS-OF-THIS-TANKER where ORDER-FOR-TANKER < 3'. LAST-TWO-INSPECTIONs and INSPECTIONS-OF-THIS-TANKER would be attributes of the same class, and the former would be a subset of INSPECTIONS-OF-THIS-TANKER.

(6) The intersection, union or difference of two multi-valued attributes can be used to define a new attribute type. An example: an OFFICER's CONTACTS is derived by taking what 'is in SUPERIORS or is in SUBORDINATES.' Intersection and difference can be used in a similar manner.

(7) arithmetic expression: An attribute can be stipulated as derived from other attributes using the operators +, -, *, / and ! (exponentiation). For example, TOP-LEGAL-SPEED-IN-MILES-PER-HOUR

may be '= ABSOLUTE-TOP-LEGAL-SPEED / 1.1'

(8) the operators maximum, minimum, average and sum can be applied to a multi-valued mapping of reals or integers. This is used to define attributes like AVERAGE-SENIORITY as 'average of SHIPS-CREW.SENIORITY'. Such an attribute will be called a "function" attribute.

(9) an "occurs" attribute can have its value set to the number of values in a multi-valued mapping, with the option of ignoring duplicates. Such an attribute will be equal to the number of (unique) values that the specified attribute has taken on. In class OILTANKER, NUMBER-OF-INSPECTIONs can be specified as 'number of members in INSPECTIONS-OF-THIS-TANKER'.

A special term which can be used in derivations is "contents". This represents the members of a grouping class. Thus for example CONVOY can have attribute OILTANKER-CONSTITUENTS defined as "subvalue of CONTENTS where is in class OILTANKERS". It should be noted that by using a combination of these derivation primitives, more complex inter-attribute relationships can be specified, if necessary.

4.2.5.2. Class Attribute Derivations.

The derivations called same, subvalue, intersection, union, difference, arithmetic expression, minimum, maximum, average, sum and count can be applied to class (as opposed to member) attributes as well. They then define a derived class attribute. For example, class attribute TOP-LEGAL-SPEED-IN-MILES is "ABSOLUTE-TOP-LEGAL-SPEED / 1.1". Two additional derivations can also be used. One indicates that a class attribute equals the

'number of members in this class' (e.g. NUMBER-OF-INCIDENTS); the other equates the class attribute with the maximum, minimum, average or sum of some member attribute taken over the whole class. An example of the latter would be class attribute TOTAL-SPILLED of OILSPILLS, "sum of AMOUNT-SPILLED over all members of this class".

4.2.6. Representing Relationships.

Attributes represent either properties (if their valueclass is a name class) or relationships (If their valueclass is a class i.e. entity type). Thus the usual method of specifying associations between entities is by means of attributes. For example CAPTAIN is an attribute of SHIPS with valueclass OFFICERS - hence it indicates a relationship between SHIPS and OFFICERS. Associations can also be explicitly defined by means of inverses and matchings, as explained in the next two sections.

4.2.6.1. Inverses.

Inverses are a means of explicitly specifying binary relationships. To show that two classes are related, each can have an attribute with a valueclass equal to the other class, these attributes being designated inverses. Class SHIPS can have an attribute COUNTRY-OF-REGISTRY with a valueclass COUNTRIES, and COUNTRIES can have attribute SHIPS-REGISTERED-HERE, valueclass SHIPS. COUNTRY-OF-REGISTRY and SHIPS-REGISTERED-HERE are inverses representing an association between COUNTRIES and SHIPS.

4.2.6.2. Matchings.

Matching enables an N-ary relationship to be specified as an entity type. This involves defining a class, say R, to represent this relationship. Participating entities can reference R by declaring attributes that 'match' to R, if these are significant. Example: the relationship between OFFICERS and SHIPS can be represented by a class ASSIGNMENTS with attributes SHIP and OFFICER (and possibly others such as ASSIGNMENT-DATE). Class SHIPS can then include an attribute CAPTAIN defined as 'match OFFICER of ASSIGNMENTS on SHIP.' This means that CAPTAIN will be assigned all values of OFFICER where the SHIP under consideration is this particular ship. By adding a third attribute such as VOYAGE to the ASSIGNMENTS class, it can represent a ternary relationship, etc. It should be noted that matching classes afford a natural way of including attributes of a relationship with the relationship they qualify, ASSIGNMENT-DATE being one such example.

4.3. Reasons For Basing ADD On SDM.

Four basic criteria were established as essential for the data model to be used in the ADD system. A shortlist was compiled, consisting of models possessing one or more of these qualities to a high degree. These were compared, and SDM chosen as the most suitable. This section describes the criteria used in model selection, and analyses the shortlist of data models to show why SDM was preferred.

4.3.1. Criteria for Model Evaluation.

In choosing a model to serve as a starting point in the database design process, the following criteria were established. Most of all, the model had to be easy to understand, remember and use. This means it had to be simple, without any complex or confusing constructs. Secondly, it had to be conducive to obtaining a complete specification, containing as much information about the application environment as possible. Further, it should be rich in semantic features, such as attribute descriptions (type, format, range) and integrity assertions. Lastly, dynamics constructs were considered undesirable. The processing perspective should play no part in creating a logical database structure, as was established in the previous chapter, since these are not inherent properties of the data and change over time. Furthermore, models which include dynamics tend to incorporate much of the semantics of the data in the processes which operate on the data. Although transactions and data volumes must be considered when refining a database to meet performance requirements, they should not determine the records or relationships in the database. Models such as EDSM [King 1981] were accordingly not considered in the final selection.

A shortlist was compiled comprising the models of HAmmer and McLeod [Hammer 1978, Hammer 1981], Smith and Smith[Smith 1977], Chen [Chen 1976, Chen 1977, Chen 1980], Abrial [Abrial 1974], Chang and Cheng [Chang 1978], Roussopoulos [Roussopoulos 1979] and Grabowski and Eigner [Grabowski 1979]. Each of these models will be considered in turn before evaluating SDM itself.

4.3.2. Data Models Analysed.

The simplicity of SHM, the model of Smith and Smith, is its most appealing characteristic. This model is easy to conceptualise (as a hierarchy of aggregates) but difficult to design. The idea of building elements by recursively combining other elements is a neat idea in theory but of little help practically. The designer has to rely on his intuition in defining the aggregates and if he works top-down there is a danger of misconception(s) permeating the model. If bottom up, he may begin with too many aggregates and not be able to recognise all the meaningful relationships between these. It is easier to list the relationships in which each entity participates, as is done in SDM, than to name all assoications and then state the entities involved. In addition, the idea of an aggregate may itself be difficult for a non-DP person to fully understand, hence an incorrect model could result. An example of an aggregate is given in fig. 4.1.

```
type ENROLLMENT = agg([P#,(C#,SEMESTER)]
  P# : PUPIL;
  (C#,SEMESTER) : KEY CLASS;
   G : GRADE
  end
```

Figure 4.1 An example of an aggregate.

The extended version of SHM contains dynamics constructs, which would have to be ignored if it were used by ADD. The model is less expressive than the others on the shortlist, as it includes very few integrity constraints. Furthermore, it was felt that

because of the difficulty of finding nouns for aggregates, and because there are too few modelling tools, one would be less likely to obtain a complete specification using SHM.

The model much more likely to produce a complete specification is that proposed by Chang and Cheng. Their database skeletons can be described either graphically or by a collection of 3-tuples (relationship.attribute, relationship, relationship.attribute). It is far easier to design such a model by building up a graph than be trying to list all relevant 3-tuples (i.e. all associations); but this leads to problems with unwieldy graphical representations. In addition, because it is based on two elements, attributes and binary relationships, the designer has to begin his model by considering far too much detail - viz. all the attributes of his environment. The wealth of relationships available also tends to involve a very large model, usually with several relationships connecting two attributes. For example, ITEM and SUPPLIER# can be connected by an FD and 'IS-SUPPLIED-BY' and its inverse 'IS-SUPPLIER-OF'. This not only makes specification tedious and time-consuming, but also provides extra work for an automatic tool using this model. An example of a database skeleton "submodel" was given in fig. 3.4. Database skeletons have a very rich set of relationships, in fact if anything too much so. Many of the functional relationships are complex and difficult for a computer-naive designer to understand and recognise in his environment. Thus although it is exceptionally semantically expressive, it was decided that SDM surpasses this model when the factors of ease of use and semantic expressiveness are considered jointly.

Other binary relationship models suffer from the same problems:

too much detail at the start, and a choice between two representations: 'awkward' graphs or 'unnatural' tuples. An example of the latter is in Abrial'S model where to avoid diagrams (or in addition to these) a relationship can be given as a 4-tuple e.g. REL (person, personofsex, sex=AFN(1,1), personofsex = AFN(0,infinity)). Since these models do not allow n-ary associations for n>2, the designer must specify more relationships than is necessary. For example instead of a relationship SALE involving a SELLER,PART,BUYER and PRICE, 4 binary relationships must be used (see fig. 4.2).

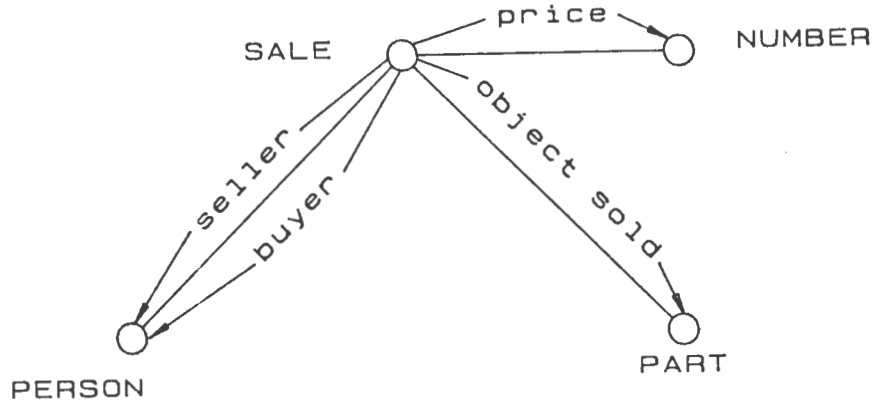


Figure 4.2 A binary relationship model of a 4-ary relationship.

The situation is aggravated by the fact that inverse relationships must be named and described. Hence there is extra time wasted by both the human and the automated design tool in processing all this information.

The binary relationship models of Abrial and of Grabowski and Eigner also have less semantic information than does say SDM. The

latter includes some properties of relationshps such as manual/optional and in-/dependent. The former contains some integrity constraints, but these are expressible in terms of operations on the data (dynamics). It would appear that this lack of semantics arises because an entity is not recognised as the basic modelling construct, as is the case with the other models on our shortlist.

Chen's model includes entities as a 3rd, separate modelling element. However it is also a graphical model and is actually restrictive in that relationships cannot themselves partake in relationships. In general, this model should be sufficiently simple for a non-DP person to use and use correctly, yet it is somewhat lacking in semantic assertions. The model of Roussopoulos which is based on concepts and frames, is superior to these models as it does not suffer from the restrictions mentioned above. However, while The idea of "pre-defined" roles is good as a guideline, the designer should not be limited to these. More important, this model is difficult to formulate and hence is unsuitable for non-DP model designers.

4.3.3. Advantages of SDM.

An SDM model comprises objects-of-interest (classes), and detail information about these objects (attributes). Hence data description is expressed in a natural and understandable manner. Having to distinguish entities, attributes and relationships,or compound and component aggregates, is not nearly as appealing. The simplicity of SDM is important. There are two kinds of complexity that can occur: structural and constraint complexity.

"the less complex a data model is, the easier it is for people to understand and use it properly" [Tsichritzis 1982]. However, oversimplification may also be a disadvantage, as there are then no mechanisms to guide the user in interpreting the data.

All of the models on the shortlist, except for SHM and SDM, could be difficult for a computer-naive user to work with, either because they are graphical and become unmanageable as they grow, or use predicate logic which many people find difficult to understand. Where properties of the application environment are stated directly rather than "built into" a graph, the specification clearly proceeds faster. Graphical models "tend to suffer from the same 'navigational' problems of record-based models" [McLeod 1980]. The advantage of SDM is that is uses very simple predicates. Interclass connections and attribute derivations are remarkably straight forward and easy for computer-naive people to understand.

It incorporates many desirable features, such as generalisation (allowing subclasses to overlap), semantic relations, inverses and integrity assertions. The principle of relativism whereby several mechanisms are provided for defining relationships, is invaluable as an aid to obtaining a complete specification from the user; this principle does not exist with the other models, except to a certain extent in database skeletons. In addition, SDM also incorporates many semantic constructs not present in the other models. The ability to assign attributes to a class as a whole, rather than to a particular member of the class, makes attribute specification extremely flexible : being applicable to entities, relationships or entity types. Attributes can have several useful properties specified such as non-null, unique, non-overlapping,

unchangeable and single- or multi-valued. None of the models characterise attributes as much as in SDM. This is important, since attributes represent the actual data values to be stored. The incorporation of important derived items in the model is an interesting and useful feature of SDM.

SDM was chosen not only because of its abovementioned qualities however, but also because it did not include any features that would be confusing for a non-DP user, such as FD'S and MVD's, and no dynamics facilities. The specification of FDs and MVDs is too complex for non-computer people and would compromise the correctness of their model. It is also unlikely that all FDs and MVDs would be identified. Furthermore, inferences made from given dependencies must be checked for validity [Bernstein 1976].

To summarise, we see that SDM was chosen as a basis for ADD because it captures as much of the meaning of the data as the other models, while remaining flexible and easy to use. In most models a trade-off between these two considerations is evident.

[Abrial 1974] Abrial, J.R. "Data Semantics", in Database Management, (eds. Klimbie, J. and Koffeman, K.), North-Holland, 1974.

[Bernstein 1976] Bernstein, P.A. "Synthesizing Third Normal Form Relations from Functional Dependencies", ACM Trans. on Database Systems Vol. 1, 1976, pp. 277-298.

[Chang 1978] Chang, S.K. and Cheng, W.H. "Database Skeleton and its Applications to Logical Database Synthesis", IEEE Trans. on Software Engineering, Vol. SE-4, January 1978, pp. 18-30.

[Chen 1976] Chen, P.P.S. "The Entity-Relationship Model: Towards a Unified View of Data", ACM Trans. on Database Systems, Vol. 1, No. 1, March, 1976, pp. 9 - 36.

[Chen 1977] Chen, P.P.S. "The Entity-Relationship Model: A Basis for Enterprise View of Data", Proc. AFIPS NCC, Vol. 46, 1977, pp. 78-84.

[CHEN 1980] Chen, P.P.S. "Entity-Relationship Approach to Systems

Analysis and Design", North-Holland, Amsterdam, 1980.

[Grabowski 1979] Grabowski, H. and Eigner, M. "Semantic Datamodel Requirements and Realization with a Relational Datastructure", Computer-Aided Design, Vol. 11 No. 3, May 1979, pp. 158-167.

[Hammer 1978] Hammer, M. and McLeod, D. "The Semantic Database Model: A Modelling Mechanism for Database Applications", Proc. of the ACM SIGMOD Int. Conf., Austin TX, May 1978.

[Hammer 1981] Hammer, M. and McLeod, D. "Database Description with SDM: A Semantic Database Model", ACM Trans. on Database Systems Vol. 6, No. 3, 1981, pp. 351-386.

[King 1981] King, R. "The Event Database Specification Model", Ph.D. Thesis, Computer Science Department, University of Southern California, Los Angeles CA, 1981.

[McLeod 1980] McLeod, D. "On Conceptual Database Modelling", Proc. of Workshop on Data Abstraction, Databases and Conceptual Modelling, Pingree Park CO, June, 1980.

[Roussopoulos 1979] Roussopoulos, N. "Tools for Designing Conceptual Schemata of Databases", Computer-Aided Design Vol. 11, No. 3, Map 1979, pp. 119-120.

[Smith 1977] Smith, J.C. and Smith, D.C.P. "Database Abstractions: Aggregation and Generalization", ACM Trans. on Database Systems, Vol. 2, No. 2, June 1977, pp. 105-133.

[Tsichritzis 1982] Tsichritzis, D.C. and Lochovsky, H. "Data Models", Prentice Hall, Englewood Cliffs NJ, 1982.

5. CHAPTER 5. THE USER INTERFACE.

The two special features of the ADD system are its automation of requirements gathering and of prototype database generation. In this chapter, the first of these is discussed.

5.1. User Interface Design Criteria.

Due to the importance of providing a good user interface for ADD, numerous articles on user friendliness were studied. In this chapter the general principles and techniques of user interface design are discussed, and then the interface created for ADD is described. Finally the implementation of the user interface program is outlined.

5.1.1. General Precepts.

There are several guidelines to follow when designing a user interface for non-computer people. The first is to "know the user". In the case of ADD, top or middle management will be involved. Hence the user will be intelligent, educated, interested, with no experience of computers, hurried, and with a low-tolerance and low-panic threshhold. The interface for such a person should be highly structured to lessen confusion. It should conform to the real world and to his habits and skills, and computer terminology and concepts should be avoided. Little or no training should be required and the data capture should encourage the user to think in the right way, selecting the correct action and moving in the right direction. The aim is to

allow the user to concentrate on what he is saying; not on how.

He should be at ease using the program. To this end, a normal/base state to which he can return whenever desirable, is comforting. Consistency is essential, as is a polite, friendly dialogue. Most of the user's tasks should be accomplished without too much work on his part. Novices prefer many small operations to one large one, as this is simpler and they can monitor their progress. It also provides them with more "closure" relief (the satisfaction of completing a task successfully).

The user should have a simple view of the system; precision and clarity are essential to avoid confusion. For example, the user must not be expected to think of too many things at once. On the other hand, there is a danger in over-simplifying a system to the extent where an intelligent user soon finds it tedious. It is advisable to test the system thoroughly with non-computer people during its design, as it is important to prevent users saying something which the program cannot understand.

5.1.2. Data Capture Techniques.

Fig. 5.1 lists the most common data capture methods. Which of these is chosen depends on the type of user and application. Sometimes simplicity, ease of use and safety are imperative (for which question-answer systems are suitable); in other situations it is important to provide for flexibility and conciseness of input. Two major decision to take when designing a user interface are whether it will be a man- or machine-initiated dialogue and whether users can give a free response or must choose from a fixed set of replies.

English-Language Dialogue

Limited English Input

Question-Answer Dialogues

Dialogue Using Mnemonics

Dialogues with Programming-Language-Like Statements

Form-Filling

Menu-Selection

Dialogue with Light Pens for Pointing to the Screen

Fixed-Panel Responses (Computer responds with one of a standard set of responses)

Action-Code Systems

Multiple Action-Code Systems

Building up a Record on the Screen

Multiscreen Menu

Telephone-Directory Technique

Multipart Menu

Multi-Answer Menu

Displayed Formats (aids user to recall command syntax)

Multiple Entry

Multiple-Format Statements

Overwriting

Panel Modification

Figure 5.1 Techniques for Data Capture.

One of the most common data capture procedures is a man-initiated command language system. Data capture programs with a command language usually involve reserved words followed by arguments. All commands must use similar relative positioning, field separators, abbreviation, etc. Problems can arise if commands are complex, or with optional fields and default values (which user does not know of or understand), with positioning of arguments and with users forgetting the commands and/or their syntax. It is difficult to choose a command set that is neither overly redundant nor overly irreducible, and one that is concise yet natural (with few constraints). Most users cannot be expected to remember lots of abbreviations, mnemonics and formats.

Because of these problems, reseachers have long been studying natural language understanding. Problems here are ambiguity, poor syntax, the importance of context and the fact that many words have several usages and meanings. This can be overcome if one's only concern is to pick up keywords and not to know their relationship in the sentence. In some systems, only keywords are recognised but users may "pad" these into sentences using other words. Unfortunately this can cause input to be completely misinterpreted unless users excercise care. Restricted grammar and/or vocabulary has been experimented with [Shen 1973, Weizenbaum 1966, Winograd 1972, Bobrow 1968, Green 1963, Gammill 1977], but also has its pitfalls: users still need to be trained and the program can misunderstand input without users being aware of this. Another solution is to allow natural language but to stipulate that only words distinguished in some way (eg. between brackets) will be recognised [Palme 1979], which is simplest for both user and program.

One of the "safest" methods of ensuring data is given logically and unambiguously is by means of a menu-driven system. If single letters are sufficient to make a selection, the process is easy to type and remember, and skilled users can omit the menu-displays by "typing ahead". There are several techniques for speeding up this process, such as listing options in decreasing order of usage, and allowing no reply (press "return") to indicate first option.

5.1.3. Enhancing User-Friendliness.

Once the basic data capture method has been decided there are several features which can be incorporated into the interface. The experienced user should be catered for wherever possible. One can allow him to short-cut through a dialogue, to abbreviate, to use a special key for duplicated values, etc. The novice on the other hand must be able to obtain adequate help during a run. Ideally a request for help should always be an acceptable response, and assistance should always be at hand. Some authors suggest that complete documentation should be available on-line. Command language systems usually have several forms of help request, for specific commands and arguments. if a help dictionary is utilised, then when explaining one concept, all words in the explanation for which help exists, should be underlined.

It is equally important for users to be able to view and alter previous input. If perusal of user's "file" is allowed, he can refresh his memory on names and meanings of objects. Viewing of individual items should be possible, and some systems provide a telephone-directory facility for instances where the exact name is

not known. Many systems allow backup by one command or re-specification of an operand in the last operation; others can backtrack by two or three entries. Another idea is to provide inverses for all operations where meaningful so that users can "undo" errors. If there are lengthy commands, it should be possible to delete (or alter the arguments of) a command while in the process of specifying it. Another approach involves having checkpoints to which user can return if necessary. The idea of an "interrupt" button is also useful here.

The dialogue of a program is where its user-friendliness is most apparent. Verbosity is tiresome, while messages so short as to be cryptic are of no use. The tone should be courteous, even servile; never annoying. Error messages must be informative rather than antagonistic and, like all displays, must prevent misinterpretation. Each display should present only one idea, and short lines are advisable. Thirty to forty characters long, as in newspaper columns, involves minimal horizontal eye movement. To summarise, the first step in designing a good user interface is to understand the potential user and decide on a data capture technique accordingly. This can be extended by providing help, review and edit facilities, which together with well-thought-out displays, should make the system easy and interesting to use.

5.2. The ADD User Interface.

This section describes the user-friendly interface provided by the ADD system. The general principles adopted are presented first, and then some of the implementation details are discussed.

5.2.1. Overview.

The extraction of an SDM model from a computer-naive user is indeed an awesome task. It was decided that this would be most successfully accomplished by commencing with the fundamental constructs and gradually introducing each modelling feature in turn. Hence this was the overall approach adopted by ADD. A novice user is first asked for entities, then their attributes, and is only introduced to subclasses and then grouping classes thereafter. Initially only the straight forward attribute derivations are presented to user, viz. same, expression and existence. At the end of the model specificaton, the others are presented one at a time, for user to consider applying to existing attributes.

The interface presented to the user is a menu-driven system. No lengthy menus are presented, these being split into a tree of menus so that user is instead given two or more simple menus, one after the other. There are never more than seven alternatives in a menu, since this is the maximum that man can deal with in one dimension [Miller 1956]. An experienced user can skip intermediate menus by supplying his final choice (bottom level) on being presented with the first menu. All menus are designed so that only the first letter is needed to identify an option.

In this way the user is given control over what is to be done, as advocated by Martin [Martin 1973]. If he does not wish this, the option 'any' is included in all menus: this causes the system to choose an option for him, according to his previous input.

The top-level menu is a choice between entering data (i.e. part of

the specification), editing, perusal, help and session termination. A second-level menu is then displayed according to the choice made. In the case of data entry, this menu allows him to specify whether he wants to enter classes, attributes, subclasses, grouping classes or name classes. (For clarity, these are called objects, properties, subtypes, groupings, and valuetypes; and henceforth referred to as O/P/S/G/V). the entering of all items (O/P/S/G/V) is done in a consistent manner, which is a slight variation of the system used by [Palme 1979]. This comprises ordinary natural language sentences in which the names of new objects (or groupings,etc) are in capital letters, to distinguish them from the other words. This caters for the naive or inexperienced user, for whom it is easy to simply punch in a description of his application environment, in natural language. the skilled user need only supply the new items' names. An attempt to 'understand' natural language was considered beyond the scope of this work. It would also be error-prone, cumbersome and unnecessary for fairly intelligent users. Capital letters were chosen instead of special symbols (as in [Shneiderman 1979]), since they are easier to type, and the add system will accept a word which starts in lower case but ends in capitals.

once all new items in a particular (O/P/S/G/V) category have been entered, the machine assumes control and requests details regarding formats, descriptions, etc. for these new items, so that a complete SDM specification results. If information requested is not known, a blank can be entered. Indeed no details need be given at all for an item, or the entire detail-specification stage omitted, if user wishes. This is because there is a danger in letting the system control an essentially creative process [Martin 1973]. As an example, when giving a subclass of a class C, the

user may realise that it is based on a property of C which he omitted to give earlier. So as not to interrupt his train of thought, the system permits a user to by-pass the specification of details. Instead, he can immediately define the omitted attribute. "QUIT" can be input at any stage to escape from whatever is being done and return to the top-level menu. An experienced user can operate in a 'command language' mode instead of menu-driven mode, by continually giving his next choice before the menu is displayed. By typing say "QUIT P" he indicates that properties are to be entered next.

At any stage, the user can obtain help by typing a '?'. This causes the program's most recent display to be expanded upon, giving a fuller explanation. This process can be repeated until no more help on a topic exists, in which case user is referred to the user manual.

To make the system easier to use, the same method is used for both perusal and editing. These allow him to scan and alter his SDM specification respectively. To edit or view any item, the user can either type its name, or an integer N. In the latter case the nth last entry is displayed. Two integers can be given if say the 2nd last to the 4th last entries are to be viewed/altered. (Where N is 1, the word 'last' is acceptable, being more natural than "1st last"). Items to be changed can first be displayed to check their current definition. No editing language is used, as these are always confusing to non-computer people. Once an item is identified, a "change menu" is presented. Two options on this allow for deleting the item and for leaving the item unchanged. The others correspond to the details on that item (eg. if item is an object, details are name, meaning, duplicates and identifiers).

To edit any detail user chooses it from the menu and is then asked for its new value, which can be nul. This process can be repeated if necessary. In this way the editing is accomplished easily and quickly.

There are three possible ways of terminating a session. One can stop and request that the session be ignored (i.e. all new data and changes abandoned), or "leave" to return and continue in another session, or "finish" altogether (meaning the specifiation is complete). If the first exit is chosen, this is confirmed to ensure data is not lost inadvertantly. If user wishes to "leave", he is first asked to fill in any outstanding details, if he has the time. He can type quit if he has to go during this stage. This also occurs if the user is "finishing", but in addition, before the program ends, he is presented with certain derivations and asked where these are relevant. These (wherein, class derivatives, recursion, subvalue, ordering, occurs, function) are not likely to have been used in the specificaton, being more 'subtle' and less natural. Hence each of these is explained at this stage, and all properties to which that derivation could be applied are flashed on the screen. He can then specify any desired derivatives.

In addition, at the end of a model specification the user is also asked whether certain functional dependencies exist. These questions are phrased "does the value of <X> always determine <Y>?" although FDs are not part of SDM, it was realised that some dependencies could be detected by the system. If user identifies any of these as true dependencies, then they can be used to improve the relation scheme and the records of the network to be generated. Hence it was decided to include this as a final stage of the user interface program. The FD-identification can be

omitted altogether, or can be done under the guidance of the database designer, if users so wish. It was decided to include this facility, once model specification is completed, for capable users, of for the designer to supervise. However it is important to realise that it is an optional component of the system. The FD detection process will be described in more detail in the next section.

Throughout the system simplicity, speed and ease of use have been of major importance. Thus, for example, all yes/no questions accept the return key (i.e. no reply) as a "no", with questions phrased so that this is always the more common response. Messages displayed by the program are concise but clear, and the length of each line is limited to 40 characters so that horizonatal eye movements are negligible. Appendix E is an example of a terminal session on the ADD system.

5.2.2. Implementation.

Data entered by the user is stored in five linked lists, one for each type (O/P/G/S/V). Each item constitutes a separate record in the relevant list. New items are always appended to the start of these lists, so that when details are to be supplied, the corresponding list is processed until a completed item is found. This organisation also facilitates the edit and perusal facilities, as the nth last entry will be nth in the list. There are several advantages to maintaining the linked lists in this way. The most recently entered information is more likely to be edited, and any undefined items to which user refers (in interclass definitions or derivations) are simply added to the

appropriate list. When details are requested for incomplete specifications these will then automatically be given attention.

The user interface program will probably be used in several separate sessions, as there is normally a large amount of data to be supplied. Thus each run starts by asking the user whether he is updating an existing specifiation or starting a new one. In the former case, the files on which the previous specification resides are read and their information stored in linked lists, so that it can be edited/displayed. The entire file is used to form the lists in each case, as it is more likely that users will forget 'old' information and hence want to "review" this. Upon normal termination, i.e. not upon a request to abandon the run, the linked lists are used to create new files, representing the updated model.

Due to the large amount of output that a user-friendly program requires, it was decided to store displays on files, rather than to include them in the program code as (numerous) WRITE/LN statements. This enhances the program's modularity, as writing is accomplished by calling a single procedure. It also facilitates extension or modification to the system, as the dialogue can be altered simply by changing this file, without any need to understand and edit the program itself. Two files are used, to ensure that they will almost always be used sequentially. Also, messages that could be repeated are included in the code. These messages are those informing user of an input error. In this way the frequency of resetting a file is so greatly reduced as to be negligible. Indeed, there is no noticeable delay in displaying of messages on the screen. The principle governing message display is essentially an extension of the help implementation described

in [Shneiderman 1979]. Any concept to explain to users is translated into two displays, the first a brief statement of that concept, the next more detailed. Thus any display can be answered by typing '?' for more information. If '?' is submitted after the 2nd display, user is asked to refer to the user manual. It is trivial to extend this to 3 or more levels, only the message files and the program constant NLEVELS need be changed.

An important aspect of the data gathering program is to verify the consistency of the model specification, to ensure that meaningful requirements have been given. Hence all data is checked for validity and consistency, as soon as it has been read. This becomes fairly complex, such as when checking inverses, certain derivations and expressions. Regarding the latter, in addition to the usual checking that parentheses pair off properly, and that two constants are never compared etc., the syntax is checked for compatibility with the COBOL IF and COMPUTE verbs (as used in DMS 1100 database procedures).

An important precaution that was taken was to ensure that the user is never confronted with a perplexing Pascal execution error. Thus all pointers are checked to be non-nil before use, all reading is by characters (converted to numbers where appropriate), etc.

The result, as test runs have confirmed, is a 'safe' and user-friendly program. This was achieved mainly because of modularity. The program is extremely modular, with procedures varying from simple reading of characters, to reading strings and texts, to those reading and checking mappings,etc. The latter module for example is called in the case of mappings as well as keys (a list of attributes) and lists of classes; and can be used to take information from one of two files, or from input or from a

linked list, and transfer it to one of two files, or a linked list, depending on the parameter settings. Entering of new O/P/G/S/V all involve calls to the same modules, with separate procedures needed only for certain detail specifications. The same module is also used for editing and perusal, seeing as the latter is a special case of the former. Since the program is highly modularised, the editing of previous data could easily be accomplished using the same procedures that are used for input.

The program was designed with user satisfaction as the primary objective. Hence details are given for items when it suits the user to do so, and not when most convenient for the program. Users can request a review(perusal) at any stage and can similarly edit the model at any time. An important consideration was that of making the specification as quick as possible to submit. Thus for example in finishing off, when the derivatives are presented for consideration, only attributes to which a derivative could legally be applied are listed. Another example is when searching for an item whose name as been given in an edit/view request. The most-recently used list is scanned first, as there is a higher probability that the item will be there. The only potentially time-consuming part of this program is the conversion of information on linked lists to files of data, and vice versa. However, as these only occur at the very beginning and end of a run, any delay will be acceptable to the user, as explained in [Shneiderman 1979].

5.2.3. Preparing the Model for Subsequent Processing.

As it was decided to present possible FDs to the user for

verification, the last task performed by the user interface program is to identify potential dependencies. How can ADD detect possible FD's? One method is to present the user with all possible combinations X -> Y, X and Y any attributes such that X union Y belongs to one class. Clearly this is impractical for data models with a substantial number of attributes, so a more reasonable method was adopted. For each key attribute K of class C, the valueclass of all single-valued attributes in C or its subclasses, is considered (as K determines these values in some context). Wherever the same two valueclasses appear together as singlevalued attributes, a potential FD is identified.

For example, if OFFICERS has key OFFICERNAME (valueclass PERSON-NAME) and CREW has single-valued attribute CABIN(valueclass ROOM-NUMBER), then any class C' having two single-valued attributes with valueclasses PERSON-NAME and ROOM-NUMBER respectively, could violate 3NF. For each such situation, the user is firstly asked if a dependency does exist or not. If it does, he is asked whether this dependency is equivalent to saying that PERSON-NAME -> CABIN. If this is so, the ROOM-NUMBER attribute in C' is redundant. However, if it represents a different fact (eg. in class JOBS, attribute CLEANER -> ROOM because each CLEANER is alloted a different ROOM to clean), then a new FD has been found.

To detect possible FD's, the following data must be known: for each key attribute K, the classes in which an attribute of that valueclass appears (which we shall call its IN-classes); and all attributes it determines, with their valueclass (which we shall term its FD-valueclasses). In-classes identify classes where an embedded FD could occur; FD-valueclasses represent the

valueclasses that a dependent attribute may have. A linked list of key attributes is created, with this information chained to each node. This is determined using recursion, since each time a class is considered, all its subclasses must also be examined, as its attributes are inherited by its subclasses. Once this has been done, each key attribute K is inspected in turn. All attributes are scanned and whenever one is found with the same valueclass as an FD-valueclass of K, and in an IN-class of K, the user is asked if an FD exists. If so, he is asked whether this is the same fact as K's valueclass -> FD-valueclass. In such a case the dependent attribute is clearly redundant, and is flagged to-be-erased; otherwise if it is a (new) FD, this fact is stored with that attribute description.

When the model specification is complete, there are still two issues to be resolved so that the SDM specification can be processed properly by the relational and network design programs. These are data aggregates and value types. Each of these will be described in turn. Data aggregates are items whose valueclass was given as some name class N, where N was defined as having more than one subfield. The list of attributes is processed and the valueclass of each is checked. If this indicates a data aggregate, then new attributes are added to the list at that point, for each subfield. These have their valueclass and size altered to represent the subfield format, and a name is generated for them comprising "aggregatename-fieldname". If this name is too long or is not unique, it is altered until appropriate. The aggregate and each of the subfields are flagged to distinguish them from 'simple' attributes. These flags are needed by subsequent programs.

The value type of an attribute stores information for the PICTURE clause of that item, if it is 'standard'; otherwise it stores the name of the class that it maps to. Thus if the valueclass is ROOM-NUMBER the valuetype becomes 'integer'; if OFFICERS, then the valuetype also becomes OFFICERS (an SDM class). The latter is clearly easy to perform, but finding the valuetype of a 'standard' item is not straight forward, because hierarchies of name classes can be specified. For example if RANKS is a subclass of integer and TOPRANKS is a subclass of RANKS, then the program must deduce that TOPRANKS is of type integer. Thus, a list is first constructed of all name classes such that each name class is preceeded by its superclass. (For example, RANKS would preceed TOPRANKS). The list commences with the four 'system' types STRINGS, BOOLEANS, INTEGERS and REALS, with their valuetypes set. Then each node in this list is visited in turn and alloted a valuetype equal to that of its superclass, which will already have been determined. Once this has been done, each attribute can be inspected and assigned the correct valuetype according to the information in this list.

[Bobrow 1968] Bobrow, D.G. "Natural Language Input for a Computer Problem-Solving System", in Semantic Information Processing, (ed. Minsky, M.L.), MIT Press, 1968, pp. 146-226.

[Gammil 1977] Gammill, R.C. and Shukiar, J.J. "An Interactive System for Aiding Management Decision Making", AFIPS 77, 1977, pp. 753-759.

[Green 1963] Green, P.F. et al "BASEBALL: An Automatic Question-Answerer", in Computers and Thought, (eds. Fergenbaum, E.A. and Feldman, J.), McGraw-Hill, 1963, pp. 207-216.

[Martin 1973] Martin, J. "Design of Man-Computer Diaglogue", Prentice Hall, Englewood Cliffs, New Jersey, 1973.

[Miller 1956] Miller, G.A. "The Magic Number Seven, Plus or Minus Two: Some Limits on our Capacity for Processing Information", Psych. Rev., Vol. 63 No. 2, 1956, pp. 81 - 97.

[Palme 1979] Palme, J. "A Human-Computer Interface for Non-Computer Specialists", Software Practice and Experience, Vol. 9, 1979, pp. 741-747.

[Shneiderman 1979] Shneiderman, B. "Human Factors Experiments in Designing INteractive Systems", IEEE Computer, Vol. 12, No. 12, December 1979, pp. 9-19.

[Shen 1973] Shen, S.N.T. and Krulee, G.K. "Solving Linear Programming Problems Stated in English by Computer", ACM Conf. Proc., 1973, pp. 299-303.

[Weizenbaum 1966] Weizenbaum, J. "ELIZA - A Computer Program for the Study of Natural Language Communication Between Man and Machine", Comm. of the ACM, Vol. 9, January 1966, pp. 36-45.

6. CHAPTER 6. CREATING A RELATION SCHEME.

The creation of a relation scheme from an SDM specification was incorporated in the ADD system. This was done because of the importance of the relational model, and the vast amount of research directed towards establishing criteria for 'good' relations. The theory of relational database design, which is based on a sound mathematical foundation, could be applied to ensure that relations are well chosen. Unfortunately it is not always apparent how to convert a data model into a relation scheme. Fortunately this did not prove to be the case with SDM. By carefully considering the meaning of each structural primitive, and through the application of normal form theory to these primitives, a method was devised for converting the model to a 2NF relation scheme. Since functional dependencies are not part of SDM, the criteria for 3NF could not be checked. The resulting scheme can be regarded as a conceptual schema for a relational database. Certain integrity constraints and physical considerations may be added, depending on the particular relational DBMS used. In this chapter the method of relational schema synthesis is described, and then the implementation.

6.1. The Eight Types of Relation.

An SDM description is translated into a number of 2NF relations in the following manner. In general all the attributes of a class constitute one relation to describe such entities. candidate keys are removed from such a relation and are replaced by artificial keys for the class. These are named SHIP#, OFFICER#, etc. and are drawn from the domain of integers. This is more practical from

the implementation viewpoint, being easier to use as file-indexes or sort keys. Space is also saved because of the extensive usage of keys in a relation scheme to denote associations between entities. As candidate keys are often large (e.g. an OFFICER may be identified by his full name), the use of an integer instead is far more economical. Secondly, if multi-attribute keys are allowed, relations that are not in 2NF could exist, and if several candidate keys occur in the same relation, there is a possibility that this relation will not be in Boyce-Codd normal form, hence a separate relation was created for each candidate key, consisting of a CLASS# column and a column for each attribute in that key.

To ensure the relation scheme is in first normal form, data aggregates are replaced by their subfields and multi-valued attributes (repeating groups) are represented by separate relations. For example, if SHIPENGINES is a multi-valued attribute of SHIPS, then it cannot be included in the SHIPS relation. A new relation of degree two is created, with columns SHIPS# and SHIPENGINES. In special cases where an attribute takes on exactly N values, $1 < N < 10$, an alternative method is used. Instead of creating a separate relation, N columns are reserved for this attribute in the relation describing that class. In this way the attribute PARENTS would become columns PARENT1 and PARENT2.

To preclude embedded dependencies, any attribute that is derived from other attribute(s) in the same relation, is removed to form a new relation. Functional dependencies that were detected in the model also constitute separate relations. The dependent attribute is removed from the relation describing its class, thus preventing update anomalies, as advocated by Codd [Codd 1970].

The implementation of subclasses accounted for using a sixth type

of relation. As explained in the discussion on SDM, a subclass C of superclass P inherits all the attributes of P. It was envisaged that the attributes of P would have to be treated as if they were included in the description of class C. However when derived, multi-valued or key attributes are considered, it is clear that separate relations C#,attribute for all such combinations would be absurd. Since there are generally several such attributes and often many subclasses, a large number of extra relations would result. Hence only attributes stored in the relation of a superclass P were duplicated in the relations representing subclasses of P, this being done by recursion. This too was abandoned on realising the following. If say OILTANKERS is a subclass of SHIPS and the latter has attribute COUNTRY-OF-REGISTRY which is also placed in OILTANKERS, then user must remember that both relations must be used to answer a query like "Which SHIPS are Swedish?". This is clearly unsatisfactory. Hence subclass relations only contain the attributes peculiar to them. A relation P#,PTYPE is created for each superclass P. The key of each entity in P would form a separate tuple of this relation, and its PTYPE value would specify the subclass to which it belongs. (If an entity were in several subclasses of P, several tuples would be stored for that entity). This was chosen in preference to the method used by Smith and Smith [Smith 1977], where a (boolean) column PTYPE is included in the P relation for each subtype of P. The main reason for this choice was that less space is taken up using my method.

Match classes and inverse relationships also constitute separate types of relation, as this facilitates the coding of the network generation program. Furthermore, such relations enhance the relation scheme by distinguishing the major binary relationships

and making them directly available to the user (without the need for projections).

6.2. Implementation.

The module that designs the relation scheme consists of three distinct stages. The first creates the data structures for storing relevant information from the specification. The second places each attribute in turn in the correct relation. In the final stage, the resulting relations are written to a file REL, which is used in the next system module, for network design. These stages will be discussed in the next three sections.

6.2.1. Data Structures.

Except under certain conditions, as described in the previous section, attributes remain in the relation for their particular class. For example COMMANDER and DATE-COMMISSIONED remain in the OFFICERS relation. Hence a tree is set up with one node for each SDM class to which is attached a list of all its attributes. Attributes that are part of a key are linked to their class node in a separate chain from non-prime attrubtes. Class attributes are stored in a linked list attached to a node labelled 'QENTRY'. They are linked to this "system class" QENTRY because they are not a property of any single entity, but of the entire database (i.e. they do not have multiplicity[McLeod 1980]). Each attribute in the tree so formed consists of six fields: column name, underlying domain name, multi-valued indicator, number of columns to occupy, and a removal indicator (to specify whether that attribute must be

removed from the relation for its class). The sixth field is a flag that denotes whether or not the attribute is involved in a matching. If it is, additional data about this matching is also stored.

In the preliminary stage another linked list is created comprising all classes (base or non-base) which have subclasses defined on them. This is used to create the "CLASS#,CLASSTYPE" relations. A list of attribute pairs representing inverse relationships is also constructed.

6.2.2. Relation Scheme Design.

Stage two of the relation scheme generator can be viewed as a decompostion of one large relation - viz. all the attributes in the SDM specification - into several 2NF relations. The properties of each attribute are examined in turn. If these properties indicate that the attribute cannot be included in the relation describing its class, then it is flagged as removed in the list attached to its class, and a new relation is created. If the attribute is prime or can have a variable number of occurences ('multi-valued'), this relation consists of "C#,attribute", where C is the class to which it belongs. The new relation consists of columns "Y,attribute" if the attribute is dependent on Y because of an FD or attribute derivation. The latter case is the most interesting. If attribute A in class C is derived from some mapping "B.D.E" (and possibly from other mappings as well) then attribute B must be in the same class as A. If B is multi-valued or prime, then it will eventually be removed from the relation for class C, hence A may remain there. If B is itself a derived

attribute, it will only be removed from the relation for C if it is derived from another attribute, say X, which is in C. In this case, A cannot remain in the C relation as it is dependent on X. (Through transitivity, X -> B And B -> A, hence X -> A). It is for this reason that an attribute is not physically deleted from the list of attributes linked to its class, but simply flagged as such. It should be noted that in the above example, A is directly determined by the value(s) of D, which will probably belong to another class C'. However, as B determines the value of D, B and A should not appear in the same relation.

Class attributes, initially all placed with class QENTRY, are handled in exactly the same manner as are member attributes, but with respect to the QENTRY relation. Thus dependent, multi-valued and derived class attributes also form a separate relation, where appropriate.

Each pair of inverse attributes consitutes a relation, hence these attributes are also flagged as removed. A list of all inverse pairs is set up at the start of the run. Thus as each attribute is considered, if its inverse field is non-blank or if it exists in the inverse list, it is treated accordingly. This caters for any cases where the inverse was specified for only one of the attributes. Each inverse pair in the list is flagged once a relation has been created for it, to prevent this from being done twice.

Similarly, "matching" classes are also handled differently from other classes. This is best explained by example. Suppose class ASSIGNMENTS has attributes SHIP and OFFICER. In SDM, all attributes of a match class that have a non-standard valueclass (i.e. not strings) are specified as part of its key. Hence SHIP

and OFFICER together would be given as the key for ASSIGNMENTS. However, if class SHIPS specifies that CAPTAIN matches to OFFICER of ASSIGNMENTS and is single-valued, then clearly SHIP alone is the true key of ASSIGNMENTS. Thus the program keeps a list of all match classes and correctly determines the keys of these classes as follows. The number of attributes with a non-standard value class is counted,say N. Then each of these classes is examined to see whether all the other (N-1) classes are matched to, as single-valued attributes. If so, the key of the match class is adjusted accordingly. Each match class also has the number of attributes in the resulting key stored with it in the list. When it is written to file REL, it is given one of two special codes meaning matchclass, one key attribute; and matchclass, several key items. These codes are used by the network generation program.

6.2.3. Output of the Relation Scheme.

A file called REL stores the relations created, with one record per column. A record consists of fields RCOLUMN, RDOMAIN, KEY and TYPEREL. The latter is used to indicate the first column of a relation, giving its type or code (eg. 2 for multi-valued attribute and class), and is zero otherwise. KEY is a boolean, which is true if that column is (part of) the primary key of the relation. Relations created for attributes with special properties (eg. inverses, multi-valued attributes) are written to REL as soon as they are detected. The other relations are written to REL at the end of the program. This is easily accomplished as the information is already stored in the various data structures. Appendix F shows the relational schema generated by ADD for the SDM example of appendix D.

6.3. Evaluation of the Result.

Examining the relation scheme synthesised from an SDM description, it is clear that many of the SDM features are 'lost.' This is because the relational model is less semantically expressive than is SDM. Interclass connection definitions and attribute derivations as well as certain properties of attributes cannot be incorporated directly into a relational model; the integrity constraints provided by the particular relational DBMS used must be relied upon for this.

It is appropriate to pause at this point to consider to what extent the relation scheme conforms to the rules of normative theory for 'good' relations. The relation scheme is clearly in 1NF, as all repeating attributes become separate relations and data aggregate items are replaced by their subfields. Since all candidate keys form a different relation, and are replaced by a 'system' key comprising one attribute, the relation scheme is also in second normal form. (Recall, this means each attribute is functionally dependant on the complete key, and not some subset thereof).

To detect violations of 3NF requires recognising a dependency of an attribute X on an attribute Y in the same relation, where Y is not the key of that relation. Within the framework of an SDM model, attributes with a derivation 'same as Y' or equal to some arithmetic expression involving Y, or 'ordered by Y' are the only instances where potential violation of 3NF can be found. Hence such attributes are transferred to a separate relation by the ADD system. The other derivations all involve multi-valued attributes, which will be placed in a different relation and hence

cannot cause any intra-relation dependencies. As was pointed out in the preceeding discussion, if X is derived from Y.A.B.C say, then it is indirectly determined by Y, and hence does not appear in the same relation as Y.

Since SDM does not include the concept of FD's, one cannot be certain whether other unspecified dependencies exist within a relation, and hence no claim can be made to produce a 3NF relation scheme. Taking this factor into account, the ADD system incorporates a process whereby the possibility of an FD is identified and presented to the user for verification. (This was outlined in the previous chapter). If such an FD is identified as having the exact same meaning as another relationship elsewhere in the database, the dependant attribute is dropped to avoid redundancy. Otherwise, the attributes involved in the FD constitute a separate relation. The former case where a redundant attribute is recognised is just as important, because if that attribute were not removed the relation would not be in 3NF.

It was decided that only keys consisting of one attribute would be considered in the FD detection process, for two reasons. Firstly, asking a user if X,Y -> Z is likely to confuse him and lead to incorrect replies, as this is more difficult for non-DP people to understand than "does X determine Z ?". There is clearly no benefit in deducing extra information about an SDM model if its validity is questionable. Secondly, even if users could understand and recognise such FD's, their presence would really only indicate that a poor SDM model has been designed. This can be deduced from the fact that such a dependency implies that name classes are being used as the valueclass of attributes, instead of classes (entities). For example if a class C has attributes X

(valueclass surname) and Y (valueclass initials), where SURNAME + INITIALS is the key to OFFICERS, then probably a relationship between C and OFFICERS has not been recognised by the user. Another interesting point is that possible FD's X -> Y are only considered for cases where the valueclass of X is a name class and not a class (object). The latter could also have been considered to discover more redundant attributes or embedded dependencies in the relation scheme. However, ADD is primarily a tool for designing a codasyl database. Attributes such as X would become sets in a network. thus it was decided not to burden the user with any more questions regarding such FD's, as this knowledge would not affect the network records.

In conclusion therefore, the relation scheme generated is in 2NF, and any violations of 3rd normal form which are detected directly from the SDM model or with the aid of the ADD system, are removed.

[McLeod 1980] McLeod, D. "On Conceptual Database Modelling", Proc. of Workshop on Data Abstraction, Databases and Conceptual Modelling, Pingree Park CO, June 1980.

[Codd 1970] Codd, E.F. "A Relational Model of Large Data Banks", Comm. of the ACM, Vol. 13, No. 6, June 1970, pp. 377-387.

[Smith 1977] Smith, J.C. and Smith, D.C.P. "Database Abstractions: Aggregation and Generalization", ACM Trans. on Database Systems, Vol. 2, No. 2, June 1977, pp. 105-133.

[Sperry 1972] Sperry Univac, "Data Management System (DMS 100)", UP 7909, Rev. 3A, St. Paul Minn., 1972.

[Taylor 1976] Taylor, R.W. and Frank, R.L. "Codasyl Database Management Systems", ACM Computing Surveys, Vol. 8, No. 1, March 1976, pp. 67-104.

7. CHAPTER 7. NETWORK SCHEMA DESIGN.

The creation of a DMS 1100 schema was implemented as a 2-phase process. In the first phase, a logical network structure is derived; in the second, physical properties are added and a complete schema is output. This can be directly used as a prototype database. This chapter commences with a description of the data structure storing the design, and the subroutine which manipulates this. It then outlines the method by which the records, sets and items of the database are determined. The refinement of the initial structure to incoporate interclass connections and attribute derivations is then described, followed by the method of assigning physical parameters.

7.1. Choosing a Data Structure.

A major decision taken when implementing the system was the choice of data structure on which to store the evolving database design. This is important because of the large amount of information that must be maintained. This includes data pertaining to the schema itself and ancillary information required by the system modules. This data had to be stored in a form that was easy to access and manipulate in a variety of ways, and that existed between runs of the various modules. It was decided to use a DMS 1100 database for storing the evolving structure, as this meets both requirements and is more versatile than the data structures of conventional programming languages (e.g. linked lists). Being easier to "navigate", it is more flexible and more information is implicitly available. For example one can easily find all the sets to which a record belongs. A database is a convenient form of storage for

the prototype as it can easily be interrogated by the database administrator using QLP (a query language) [Sperry 1972]. Furthermore, being completely general, it can be used by other automated tools eg. load-program generator or query-formulation assistant. It should be noted that the database storing the schema and related information can also be used as a dictionary during the lifetime of the database. It includes the annotations (item/record descriptions) given in the SDM specification as well as item formats, integrity constraints etc.

The database storing the evolving structure is called "QDB". This contains one record type for each construct in a database. Thus there are records QSET, QREC, QMEMBR, QITEM, etc. All identifiers used by ADD start with a "Q", so that the likelihood of the same word being used in the SDM specification is minimised. Constructs that are related to each other are linked by sets, eg. QRCITM links each QREC to the QITEMs it contains. From the illustration of the QDB in figs. 7.1 and 7.2, it is clear that this is capable of storing a large amount of inter-related information, which could not easily be accomplished using the data structures of programming languages. The most interesting part of the QDB database is that which stores how the records are interlinked. Each QSET is linked to the owner's QREC via a QOWNS set occurrence. Then one QMEMBR occurence is stored below it, for each member type in the set, forming a QSTMEM set. A QMEMBR contains a description of the membership it represents (e.g. automatic/manual) but not the name of the member. This can be found through its membership in another set type, QRCMEM, which has owner QREC, and represents all sets that this QREC participates in as member.

Storing the member record name in a QMEMBR record proved

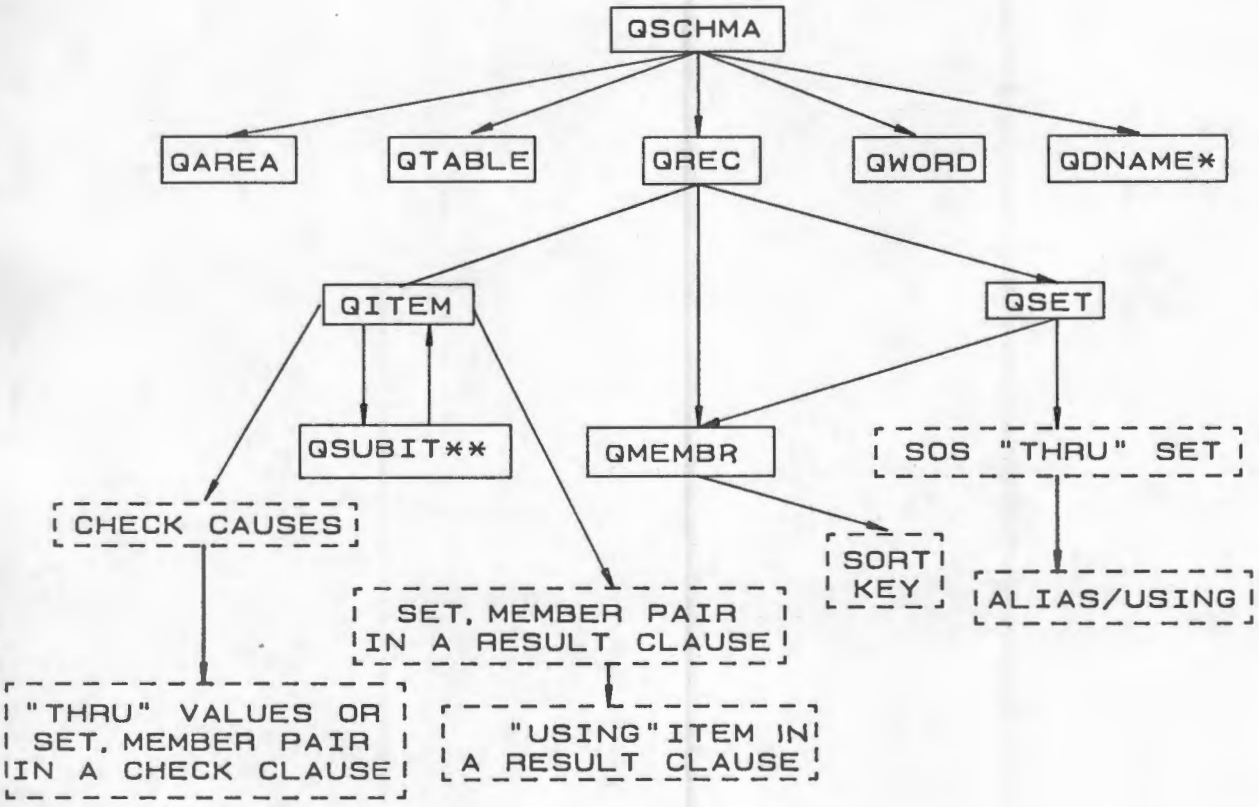


Figure 7.1 Simplified Diagram of the QDB Structure.
Records indicated by broken-line rectangles do not show their QDB names, but their meaning (purpose).
* Data Names stored here.
** Subitems (subfields) represented using these records.

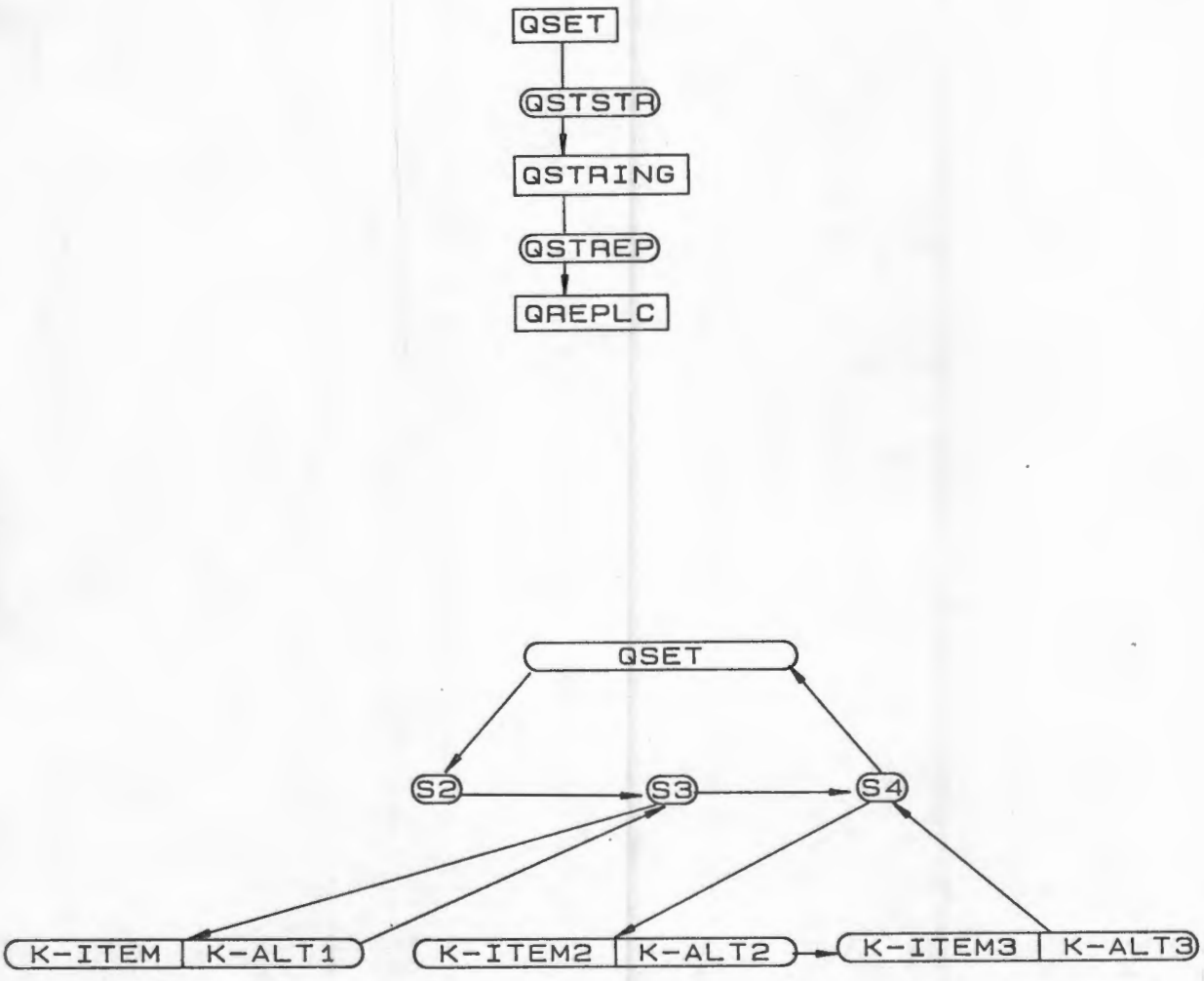


Figure 7.2 Section of the QDB showing how a SET SELECTION clause for a multi-level path is represented. QSTSTR stores the "THRU" sets in the clause; QSTREP stores ALIAS or USING items for each of these. The above example represents "SET SELECTION IS THRU S2 USING CURRENT OF SET THEN THRU S3 ALIAS FOR K-ITEM IS K-ALT1 THEN THRU S4 ALIAS FOR K-ITEM2 IS K-ALT2, ALIAS FOR K-ITEM3 is K-ALT3".

unnecessary, as fetching its QREC owner is trivial, and in this way space is saved, as there will generally be many QMEMBR occurrences. Another interesting part is that which stores data aggregates and their subfields. This was recognised as a "loop" relationship between an (aggregate) item and its (constituent) items. Hence two sets QIS (QITEM -> QSUBIT) and QDAGG (QSUBIT -> QITEM) are used to store this information, as advocated by Date [Date 1981]. (See fig. 7.3).

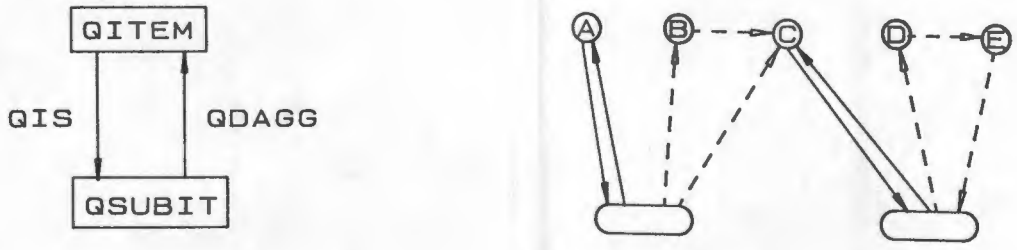


Figure 7.3 Section of the QDB showing how data aggregate items are represented.
In this example, item A is composed of two items B and C, and the latter comprises two items D and E. Solid arrows represent QIS sets, broken arrows represent QDAGG sets.

7.2. Database Manipulation Subroutine.

To manipulate the QDB database the Pascal design programs call on a Fortran/DML subroutine. It is because the Fortran DML is used that all names used in the QDB are only 6 characters in length.

The parameter settings indicate which of several possible operations is to be performed. Most of these are basic - they store/find/delete/modify some record type. Others are more complex, moving all items from one QREC to another, changing a QREC's set membership, etc. The error checking mechanism of DMS 1100 is fully utilised to prevent undetected errors occuring. "ERROR-STATUS" is tested after each DML command, and if anything is amiss, an error message is written out identifying the error (the record, set, area involved as well as error function, error number, rollback error code) and the statement at which it occured. In such a case a parameter IERROR is set true, to inform the caller that the operation was not executed, but the program is not terminated. IERROR is also used as an indicator in normal conditions, such as with calls that find out if a QREC exists.

Another usage of IERROR is in checking whether a name (identifier) has already been used in the prototype database. This is imperative since all record names, set names, etc. must be unique. For this purpose, a QWORD record type exists and a new occurrence of this is stored each time a new name is used. All such names are restricted to 6 characters so that a Fortran subschema can be generated easily, without needing to replace schema names by Fortran versions. The original name, as given in the SDM specification, is also stored, and is checked whenever the database is used. (Both the 6-character and original names are passed as parameters where applicable). The 6-character names are formed in the Pascal programs by taking the original and dropping firstly vowels (working from the end to the beginning of the word), and then consonants (in the same direction) until fewer than seven letters exist. Thus "SHIPS" stays "SHIPS", "OFFICERS" becomes "OFFCRS", "COMMANDER" becomes "CMMNDR". Although this may

not always give the best result, it is adequate in almost all cases and has the added advantage that knowing the original name it is easy to work out its 'Fortran' version. Flag items (described in sections 7.4 and 7.5) are formed from the first six letters in the original name, so that the name generated is not one already in use.

In conlusion, it should be noted that the QDB together with the Fortran/DML subroutine, which does one of 26 different database manipulations, can be regarded as a data dictionary and a set of operators for accessing and updating it.

7.3. Schema Creation.

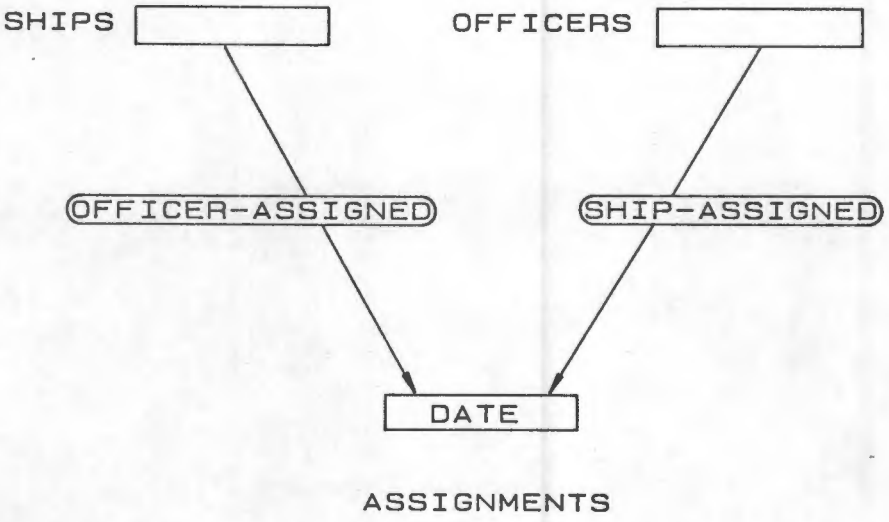
7.3.1. Relation Scheme Conversion.

The network design program takes each relation from file REL and adds its attributes to a network structure. A relation has a code number indicating its type, and is dealt with accordingly. Relations representing a class and its attributes become a record in the database with these attributes as fields. Prime attributes are treated in the same way as non-prime ones. The details of this process are given in the next section (7.3.2). Relations constructed for a class and its multi-valued attribute translate to a set with a member record representing the attribute. This member comprises one field, in which a value for that attribute can be stored. This method is suggested as the best way to implement repeating attributes by authors such as Taylor and Frank [Taylor 1976]. Relations representing inverses are translated

into set(s) according to the key of the relation. If only one of the two attributes is prime, then its valueclass becomes the member and the other valueclass becomes the owner of a set. For example, the inverse pair COUNTRY-OF-REGISTRY, SHIPS-REGISTERED-HERE has key SHIPS-REGISTERED-HERE, since a SHIP cannot have more than one COUNTRY of registry. Hence a set COUNTRY -> SHIP is created. If both of the attributes are prime, then a many-to-many relationship exists, and so a dummy LINK record is created, and becomes a member of two sets, one with owner COUNTRY and the other with owner SHIP. This LINK record has a unique name generated for it.

Relations that represent a "match class", i.e. a class referred to in some matching, are handled according to the number of prime attributes. A match class becomes one set, or several sets with the identical member record type, which we shall call M. Non-prime attributes with a standard valueclass (strings) become fields of this member record; while other non-prime attributes each become a set with the record that represents their valueclass as owner, and M as member. Generally a record is created to represent the "match class" itself, and this is the member M. However, if there is only one attribute in the match class key, say a, this means that a is associated with only one occurrence of each of the other attributes. Hence, if the valueclass of a is not standard, then the record created for its valueclass is used as M and no new record is needed (see fig. 7.4).

Relations representing a functional dependency X -> Y are translated into a set, so that the records of the network do not contain embedded relationSHIPS. A new record is created to act as owner, consisting of two fields, in which a value of X and the



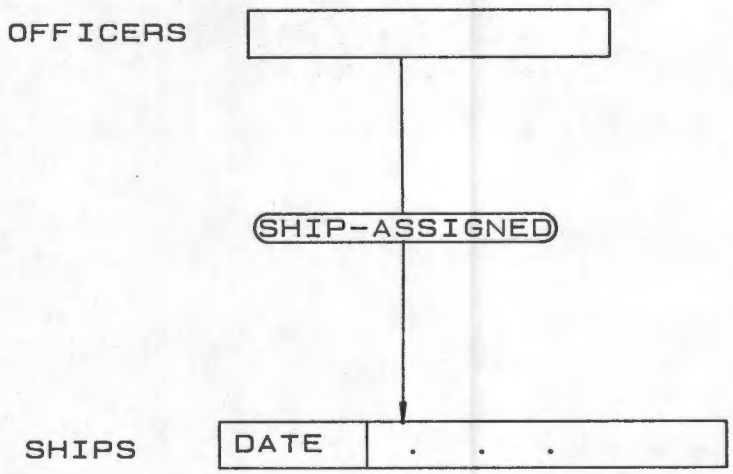


Figure 7.4 Illustration of match class representation. In the upper diagram, the key of ASSIGNMENTS is <OFFICER-ASSIGNED,SHIP-ASSIGNED>. In the lower diagram, the key is <SHIP-ASSIGNED>.

corresponding value for Y can be stored. The record which Y qualifies becomes the member, as it is associated with only one X (and Y) value. The remaining kind of relation, the "C#,CTYPE" kind, is not considered by the network generation module, as it uses the SDM description to resolve interclass connections. While relations are being converted to fields, records or sets, a list is also kept of how these classes and attributes are implemented. This is needed for the subsequent stages of network generation, when this initial structure is adjusted.

7.3.2. Distinguishing Sets and Items.

The majority of sets in the network are derived from those attributes whose valueclass (domain) is not a "standard type", but indicates another class in the database. Thus throughout the network generation module, before an attribute is placed in a relation, its domain is inspected, and if non-standard, a set is created instead of a field.

A class has "CLASSNAME#" as column name and "integer" as domain; while an attribute is represented by "attributename" and valueclass, respectively. Examples: SHIPS#,INTEGER for the key of class SHIPS; CAPTAIN,OFFICERS# and HULLNUMBER,INTEGER for attributes. Thus when handling attributes, (where there is no '#' in the column name), the first step is to examine the domain for a '#'. If none exists, the attribute becomes an item in some record. If one is found, the "#" is dropped and what remains identifies the valueclass of the attribute. In such a case set(s) are required to link the corresponding record to the network. For example, if CAPTAIN,OFFICERS# is encountered in the SHIPS

relation, then a set SHIPS -> OFFICERS is created if CAPTAIN is single-valued. Otherwise two sets SHIPS -> LINK and OFFICERS -> LINK, are generated. Note that OFFICERS -> SHIPS cannot be used, as there is no way of telling whether the OFFICERS:SHIPS mapping is "to one" or "to many". To avoid duplication of SHIPS, the latter is always assumed.

In creating sets, "loops" must be detected and an alternative implementation used, so that the database created will be compatible with DMS 1100. The same record type may not be both owner and member of a set type. In such a case two sets are used, in the manner advocated by Date [Date 1981]. A LINK record is the member of one set and the owner of the other. For example, if COMMANDER is an attribute of OFFICERS whose valueclass is OFFICERS, the sets constructed would be "is" (OFFICERS -> LINK) and "commands" (LINK -> OFFICERS).

Whenever an attribute is encountered in REL, its SDM description is examined to discover its chacteristics. In this way, when a field is added to the network, all the details for a full DMS 1100 item description are supplied (PICTURE,USAGE etc). For attributes represented not as fields but as sets, the "exhausts valueclass" property determines whether the set must have an automatic or manual member. In a "true" CODASYL database, the "cannot be changed" property would similarly decide the retention clause (not included in DMS 1100).

7.4. Interclass Connections.

SDM incorporates the definitions of subclasses and grouping classes. Before explaining how the schema is adjusted to

accommodate these, the integrity constraints avaliable in DMS 1100 must be examined. This is necessary because these interclass connections have integrity assertions associated with them which should be incorporated into the network database.

7.4.1. DMS 1100 Integrity Facilities.

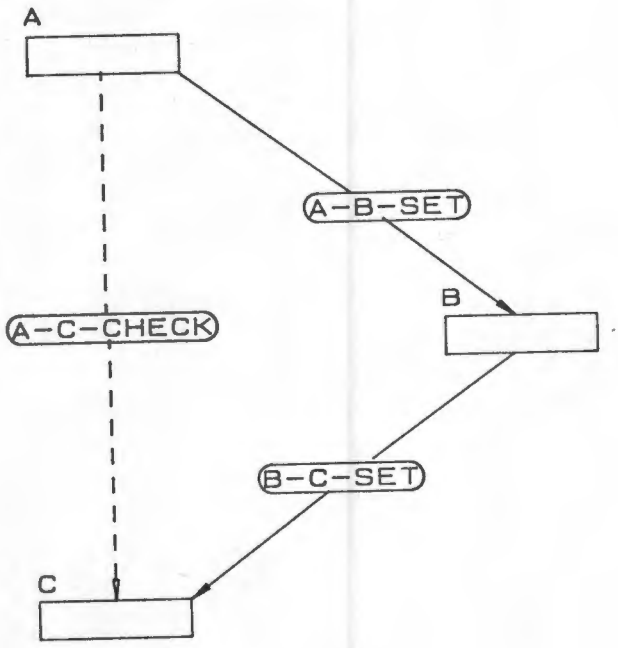
DMS 1100 includes facilities for specifying integrity constraints at the record, data aggregate and data item levels [Sperry 1972]. There are two clauses which can constrain the value of an item, viz. CHECK and RESULT. The CHECK clause allows for specification of the type of value and/or range of values an item may assume. Whenever an item with a CHECK clause is stored or altered, the DBMS enforces that constraint by returning an error condition, instead of performing the operation, if the value is not permissible. CHECK IS DATA-TYPE will check that the value is of the correct type, CHECK IS PROCEDURE P will cause database procedure P to be called to determine the validity of the value (e.g. a multiple of 10 or a valid date). The minimum and/or maximum value, or range(s) of permissible values, can also be stipulated as well as (NOT) NULL. This clause is clearly unsuitable for most of the attribute derivations and interclass connections, as they require checking an item's value against that of other database items. Hence an alternative method of ensuring the integrity of the generated database was sought.

The RESULT clause, when specified for a data item, causes the DBMS to set the value of that item when the record it belongs to is stored or modified. This item cannot be given a value by an application program. RESULT can specify that the item be a) equal

to the sum of, or some function of, other items in the same record or b) equal to the number of members of any set of which that record is the owner or c) equal to the sum of, or some function of, items in set(s) owned by that record. Where the function is not summation, a database procedure must be written. From this it can be seen that an item I, whose value depends on that of some other item(s), can be set by the DBMS if these other items are in the same record as I or are linked to it by set(s). The record to which I belongs must be owner of these sets. This was the only way in which subclasses and derived attributes could be correctly maintained.

On considering the syntax of RESULT, it was first envisaged that any derivation could be incoporated by simply connecting the two related items by means of a set. However, there is no way to ensure that this set is correcly used: that is, the user could link an owner to the wrong members. This problem is illustrated in fig.7.5.

Another approach was subsequently tried to 'extend' the applicability of a RESULT clause. If a set R1->R2 was required and R2->R1 existed, it was thought that a duplicate of the relevant R2 item could be made in R1. The required RESULT clause could then operate "ON THIS RECORD". The original item could be RESULT of procedure QEQUAL using the new item in R1. However, not every R2 occurence will necessarily have R1 records associated with it; that is, some R1->R2 sets may have no members. In such a case, there would be no way of initialising the item in R2. Hence this plan also had to be rejected. Accordingly, a RESULT is only specified if items are in the same record, or are already connected by an existing set which has the required owner and



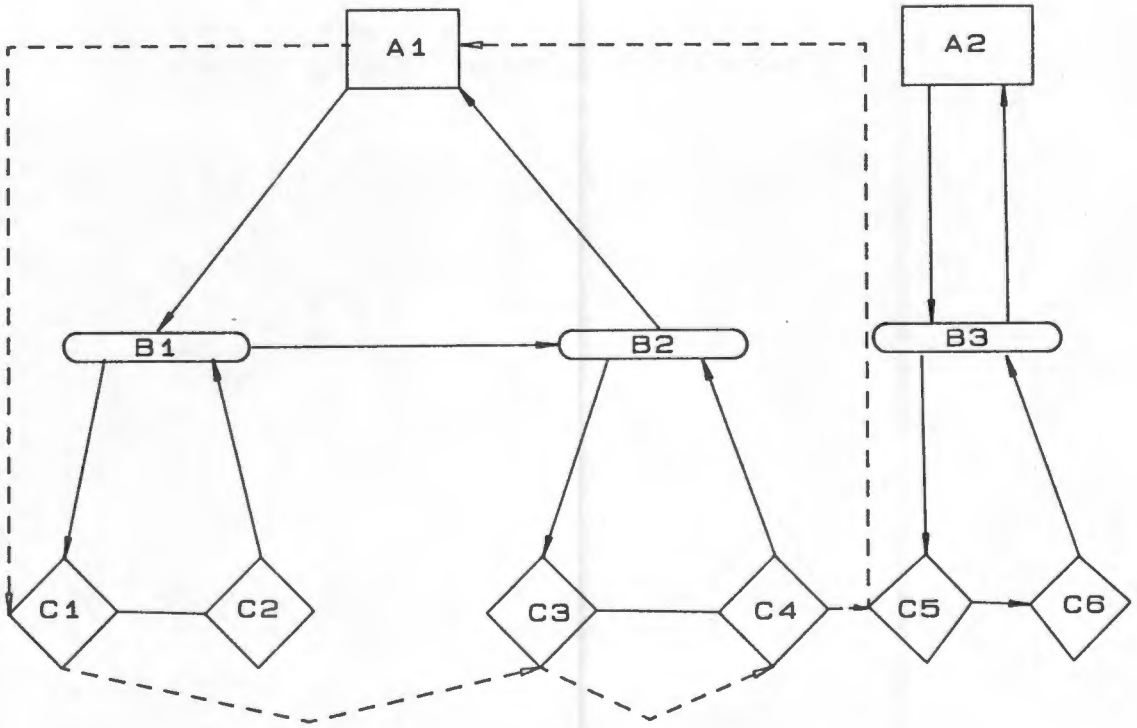


Figure 7.5 An example showing that sets cannot be added for the sake of a RESULT clause.
Here A1 is connected to C5 in an A-C-CHECK set occurrence, instead of C2,
and this is incorrect as C5 is not related to A1.
Also, C5 and C6 have not been linked to A2.

member records.

7.4.2. Subclass Connections - Design Philosophy.

This section firstly discusses how subclasses are implemented, and why. Grouping classes and attribute derivations are discussed in succeeding sections. If OILTANKERS is a subclass of SHIPS, then in addition to their own attributes, OILTANKERS have all the attributes of ordinary SHIPS. As most SHIPS will not be OILTANKERS, it is inefficient to declare a SHIPS record to contain both SHIP and OILTANKER attributes, as the latter will be null in most instances. It is also dangerous to treat SHIPS and OILTANKERS as completely separate entities, because then if user asks for all SHIPS in the database, he must know to list all OILTANKERs too; also all relationships in which SHIPs participate would have to be duplicated for the OILTANKERS record type. Thus separate record types are needed, but they must be connected in some way. This could either be as SHIPS -> OILTANKERS or as OILTANKERS -> SHIPS (with SHIPS being a manual member). The following should then be checked: (1) an OILTANKER can only exist in the database if it is linked to 1 and only 1 SHIPS occurrence. (2) a SHIPS record cannot have more than one OILTANKERS record linked to it (3) an OILTANKER can only be linked to a SHIP if that SHIP fits the criteria set down in the OILTANKERS definition (4) an OILTANKERS occurrence must be linked to the correct SHIPS occurrence.

After considering many different schemes, it was discovered that only one method exists to ensure that (1) to (3) above are verifiable. There is no way to check that (4) is correct, any

more than one can check whether the right OILTANKER is connected to an INSPECTION, etc. Verifying (1) to (3) involves using three flags per superclass, which we shall call C,D and M. M is an item "RESULT COUNT MEMBERS OF SET SHPLTN CHECK VALUE MAX 1"; D is a flag "RESULT PROCEDURE P", which is 0 if the SHIPS record does not meet the criteria set down for OILTANKERS, else greater than one; C is the check flag, which is "RESULT SUM ON THIS RECORD USING M, D CHECK VALUE NOT 1".

The fact that OILTANKERS is an automatic member of SHIPS -> OILTANKERS means (1) above will be adhered to, while M checks condition (2) and C and D together ensure that (3) is always true. Fig. 7.6 illustrates how this ensures the correct handling of subclasses.

M-Flag	D-Flag	C-Flag	CHECK	Meaning
0	0	0	ok	the SHIP is not an OILTANKER
1	N (>1)	N + 1	ok	the SHIP is an OILTANKER
0	N (>1)	N (>1)	ok	the SHIP is an OILTANKER, but its OILTANKER member has not yet been stored.
1	0	1	ERROR	the SHIP is not an OILTANKER but user is attempting to store an OILTANKER member below it

Figure 7.6 Table showing how the three flags enable subclasses to be checked.

To see why condition (4) cannot be checked, consider the following. The only way to do so would be to duplicate the key for a superclass in its subclass, and check these are equal. This cannot be done because a RESULT clause must always be attached to the owner record (e.g. SHIPS), not the member. Since many SHIPS

will not have OILTANKERS members, their key could not be initialised. If subclasses could be implemented as OILTANKERS -> SHIPS, (4) could still not be verified. This is because the set would have to have SHIPS as manual member, and hence OILTANKERS would be stored before being connected to their SHIPS counterpart. Thus (1) would no longer be enforced.

Finally, it should be noted that ADD detects subclasses with no attributes of their own. These do not constitute separate record types, as there is no advantage in doing so. They are simply implemented as one flag (corresponding to D) in the record representing their superclass. A tree structure is set up to store the superclass-subclass hierarchies in the model, so that the flags are placed in the correct records. Thus if A is a subclass of B, which is a subclass of C, and neither A nor B has attributes of their own, then the flag for both A and B is stored in C.

7.4.3. Checking Subclass Definitions.

This section discusses the PROCEDUREs which set the D flag; that is the procedures which determine whether a record belongs to a subclass or not. The PROCEDURE varies according to the type of subclass we are dealing with. If this is user-controllable no procedure is required; the flag is set by application programs. If intersection, union or complement, the subclass is handled as illustrated by the following examples. If BANNED-OILTANKERS is BANNEDSHIPS intersected with OILTANKERS, the BANNED-OILTANKERS flag is set to zero by the procedure if either of the flags for the other subclasses is zero. If SHIPS-TO-BE-MONITORED is "BANNED-SHIPS or OILTANKERS-REQUIRING-INSPECTION" then the

SHIPS-TO-BE-MONITORED flag is simply the sum of the other two flags. If SAFESHIPS is "not BANNED-SHIPS", the procedure sets the SAFESHIPS flag to zero if the other flag is greater than one, and vice versa. Usually all three flags will be in the same record. If not, the flag cannot be verified because of the limitations of the DMS 1100 RESULT clause. (It is impossible in a meaningful SDM model for one flag to be below the others, hence a set cannot be used in the RESULT clause).

To illustrate the "existence" connection, we shall consider a subclass DANGEROUS-CAPTAIN of OFFICERS defined as "where is a value of INVOLVED-CAPTAIN of INCIDENT". If OFFICERS -> INCIDENT exists then the DANGEROUS-CAPTAIN flag equals 1 more than the number of members of that set. (Recall that any value exceeding 1 implies true). If however the set INCIDENT -> OFFICERS exists, this is converted to 2 sets INCIDENT -> LINK and OFFICERS -> LINK, and then the DANGEROUS-CAPTAIN flag can be set as before, using the OFFICERS -> LINK set.

The most common subclass connection is that involving a boolean function expressed in terms of properties of the superclass. Consider RURITANIAN-SHIPS described as "where COUNTRY-OF-REGISTRY equals 'RURITANIA'". Here a RESULT clause can be attached to that flag if all the arguments in the expression are in the same record, or in a record connected to this as member of some set. This RESULT clause uses a special database procedure, which has to be generated by ADD according to the expression in the subclass definition. A file DBPFILE stores the relevant information for the database procedure generation program, in terms of the records and fields involved. The PROCEDURE DIVISION of a database procedure essentially consists of a COBOL IF statement corresponding to the

expression.

7.4.4. Grouping connections.

Grouping classes are generally represented by a set with the grouping class record as owner and grouped class as member. In this way each set occurrence represents one group. Examples of grouping class implementations are given in figs. 7.7 and 7.8. If the grouping connection is user-controllable , the records are linked to each other by two sets with a common member, as for many-to-many relationships (fig. 7.7a). This is because there is no way of telling whether an entity in the grouped class can belong to more than one grouping. To preclude data duplication, it is safest to assume this is possible. (Example: CONVOYS of SHIPS).

If the grouping is "expression-defined", this is handled as follows. Suppose SHIPTYPES is a grouping of SHIPS on common value of their attribute TYPE. IF TYPE is a single-valued, standard attribute of SHIPS, then a set shiptypes -> SHIPS is included in the database. A TYPE item is included in the SHIPTYPES record, RESULT OF PROCEDURE P ON MEMBERS OF shiptypes-SHIPS "CHECK VALUE NOT NULL". The procedure P checks that all members have the same TYPE value, returning this value if they do, else returning null. If TYPE were a multi-valued, standard attribute, then it would be stored in a separate record "below" SHIPS. In that case, this record becomes the member of a SHIPTYPES-SHIPS set. If the attribute is non-standard, then the grouping class record is inserted between the two records, as shown in fig. 7.8. If the attribute is single-valued and standard but is not part of the

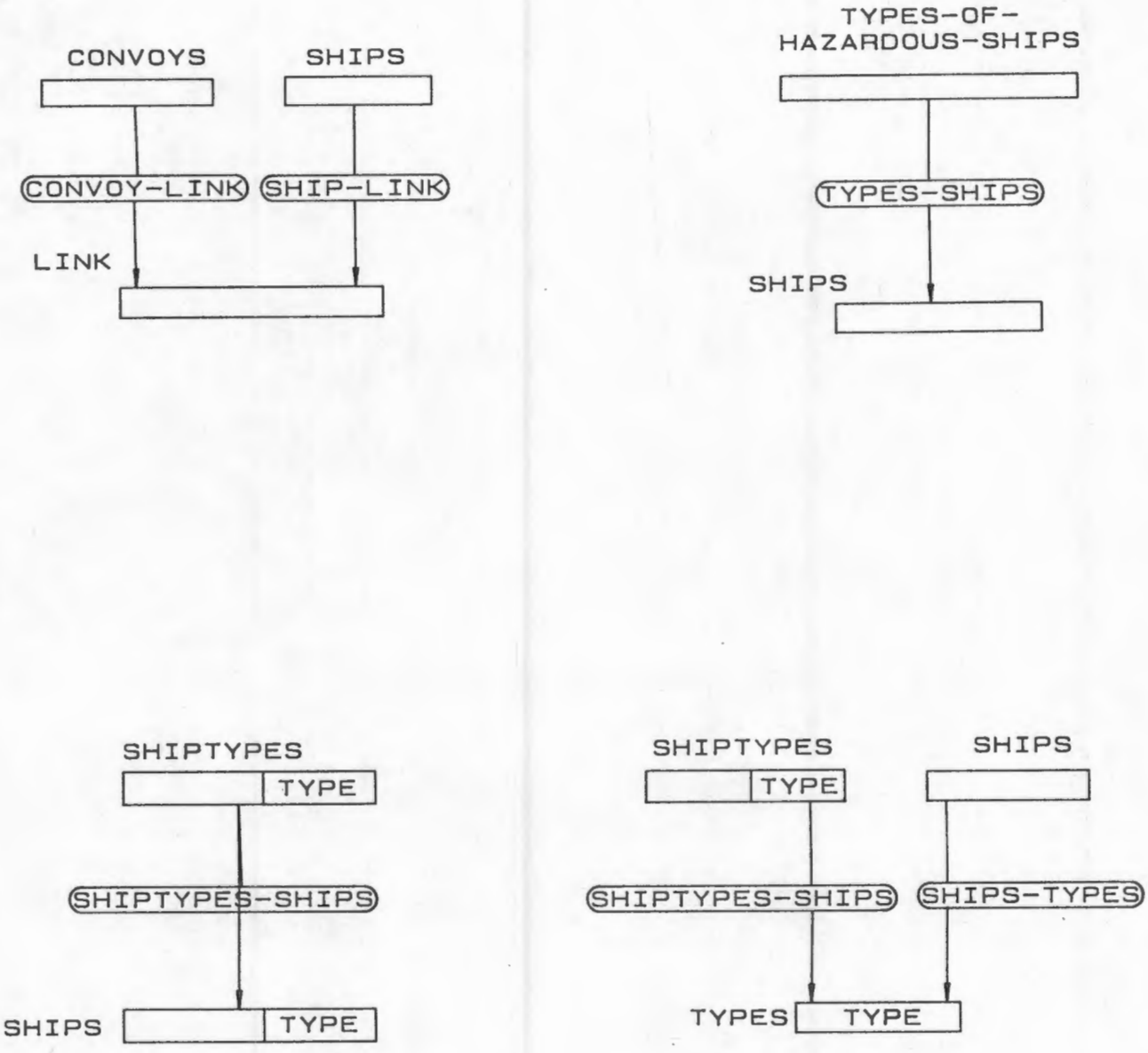
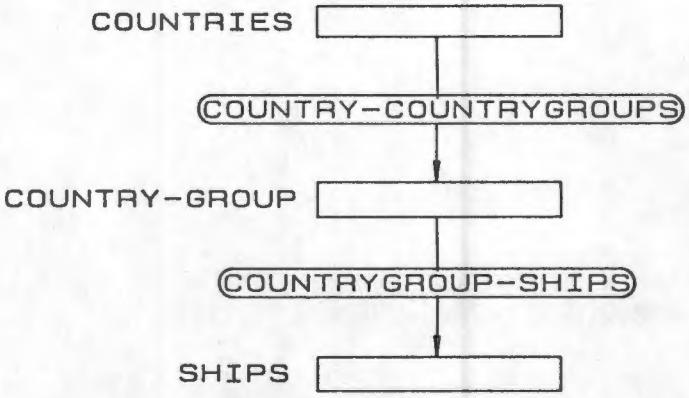


Figure 7.7 Implementation of grouping classes: some examples.



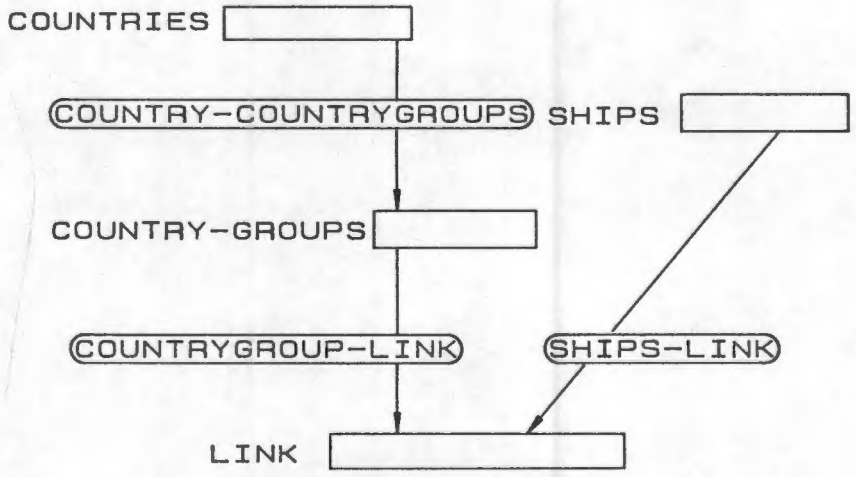


Figure 7.8 Implementation of grouping classes: more examples.

record concerned (eg. SHIPS) then the grouping cannot be checked by the DBMS as the attribute will be "above" that record in the network. However, this can only happen if the attribute was part of an FD, and thus will be a rare occurrence.

If several attributes are specified in the grouping definition, then if these are all single-valued, standard attributes they are each handled as described above. If one of them is multi-valued, then the other attributes are stored in the same record, say R, as is the multi-valued attribute, so that they can all be used to check the grouping. (The set constructed has owner grouping-class record and member R, as before). To keep the single-valued attributes in the record where they were originally, the original items are retained, but with a "RESULT OF PROCEDURE QEQUAL" on the set connecting this to their new record. Procedure QEQUAL ensures that these attributes have the same value in the members as in the owner. If several multi-valued or non-standard attributes are listed for a grouping, no more than one of these can be used to verify the grouping. This is because too many record types are involved. In such a situation the most efficient of these attributes is chosen to be checked.

We lastly consider enumerated grouping classes. Suppose TYPES-OF-HAZARDOUS-SHIPS is a grouping of BANNED-SHIPS, OILTANKERS-REQUIRING-INSPECTION and SHIPS-TO-BE-MONITORED. In such a situation a flag is added to the SHIPS record, RESULT SUM all the other relevant flags if they are in the same record (as is generally the case). A TYPES-OF-HAZARDOUS-SHIPS record is made owner of a set linking it to SHIPS, and contains a single field with a RESULT of procedure P on that set "CHECK VALUE NOT 0". Procedure P returns value zero if any member has a

TYPES-OF-HAZARDOUS-SHIPS flag with value nought. Although this ensures that no "Non-hazardous" ships participate in any such grouping, it does not constrain the formation of groups. The latter is not possible, because the groupings are determined by different (flag) items.

7.5. Attribute derivations.

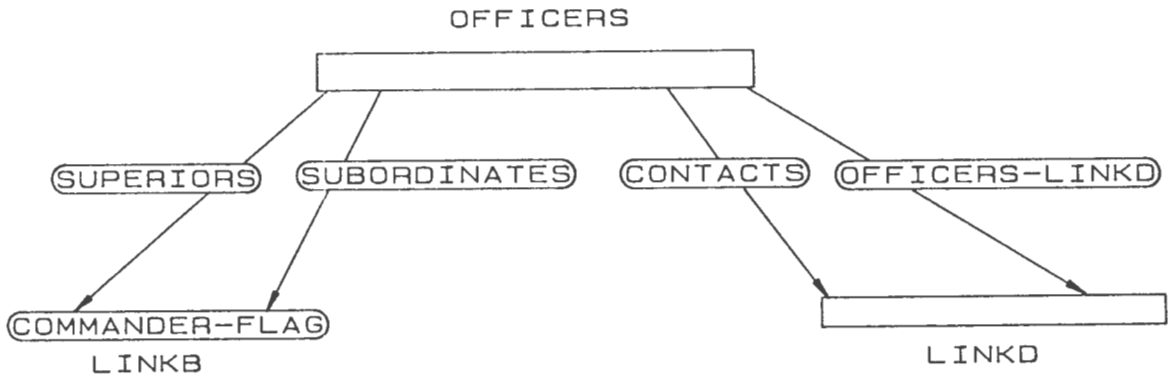
In certain cases the knowledge of what a derivation means enables the sets and records of the database to be modified. IN other cases, a RESULT clause is specified for an item so that the DBMS can derive the correct value for it.

7.5.1. General Principles.

The implementation of each type of attribute derivation will be described by means of an example.

An attribute can be specified as the union, intersection or difference of two multi-valued attributes. These three attributes would normally be implemented as three separate many-to-many relationships. Because of the derivation however, this is replaced by one member record, connected to all original owners. This member has three flags, one with a RESULT clause USING the other two flags. As an example: if CONTACTS is "where in SUPERIORS or in SUBORDINATES", the (single) member record has SUPERI, SUBORD and CONTAC flags, the latter being "RESULT SUM ON THIS RECORD USING SUPERI, SUBORD". This is illustrated in fig. 7.9.

"Subvalue" attributes are handled as follows. If attribute MAJOR-INCIDENTS of SHIPS is "subvalue of INCIDENTS-IN



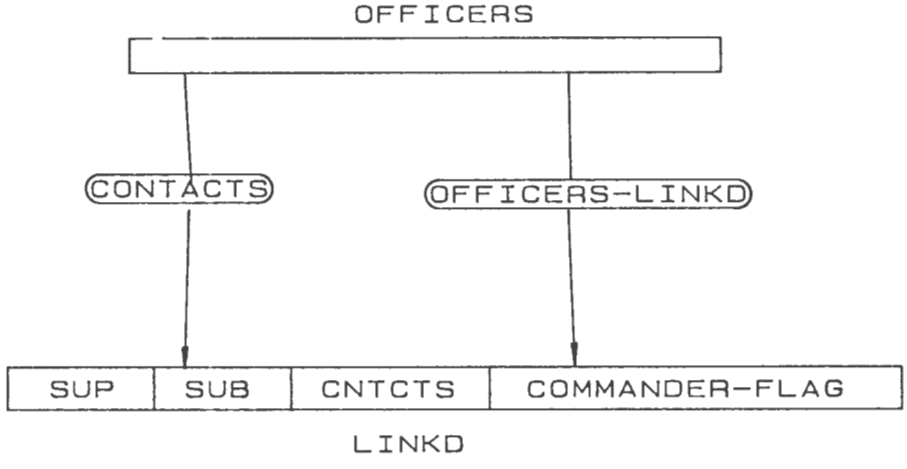


Figure 7.9a) An example where a "union" derivation is handled. The upper diagram shows the network before dealing with the derivation, the lower shows the network after handling this.

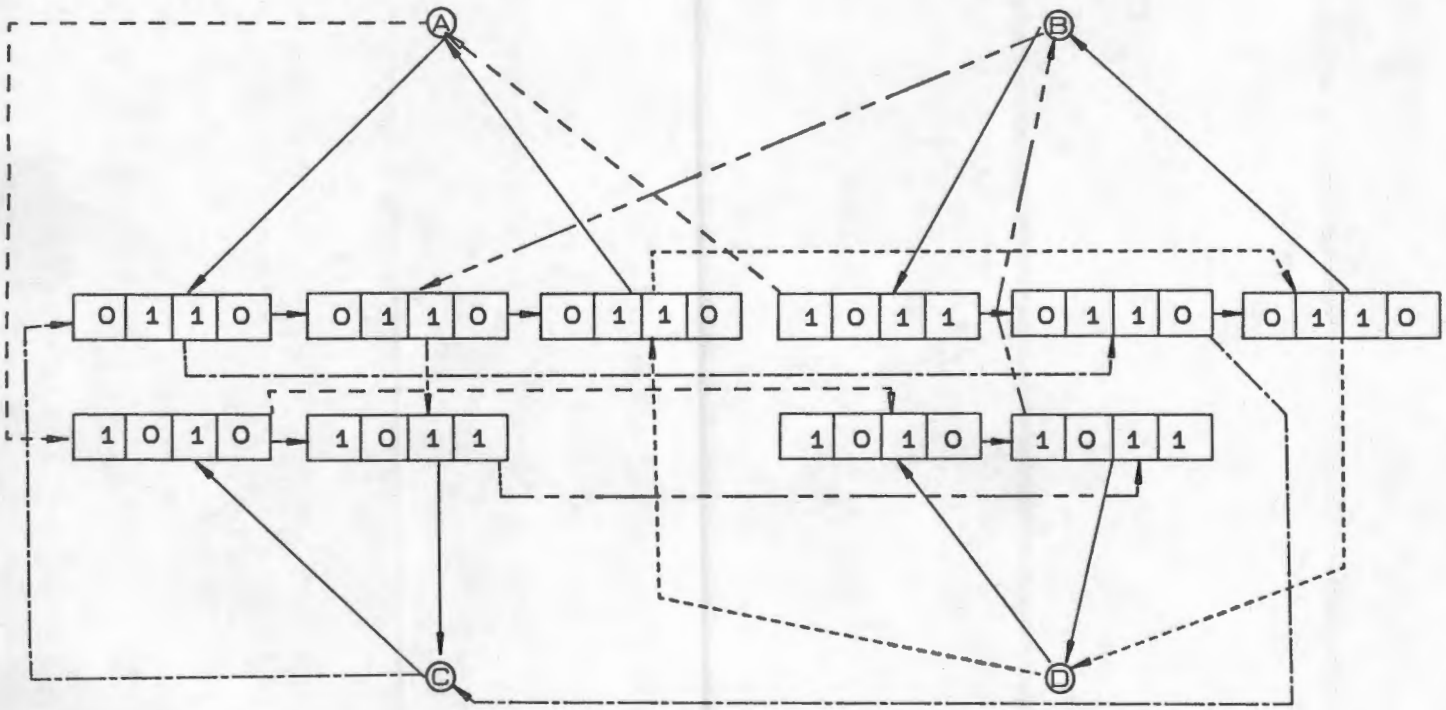


Figure 7.9 b) An example where A commands B, and B commands C and D, showing how the CONTACTS attributes are implemented.

where CODE = 1", then MAJOR-INCIDENTS becomes a flag in the LINKA record (INCIDENTS-IN is implemented as a set SHIPS -> LINKA) with "RESULT PROCEDURE P USING CODE", if CODE is in LINKA. The set is "sorted" so that all occurences of the "subvalue" attribute precede the other records. It should be noted that three or four sets would be necessary if the subvalue derivation were not stipulated (see fig. 7.10). It was first thought that one could have a LOCK on the OILSPILLS record saying "LOCK FOR INSERT INTO MAJOR-INCIDENTS IS '1' KEY IS CODE". However in DMS 1100 this "KEY" is not a field of the record, but an item of WORKING-STORAGE. Hence this facility could not be used for this purpose.

It should be noted that the expression in the subvalue can be much more complex than "CODE = 1": it can contain several items all over the network. Hence several sets may be required, to ensure that the flag is correctly set. The expression is written to DBPFILE for use by the procedure generation program, along with a unique procedure name.

If the expression is "where is in class C", then the flag indicating the derived attribute is treated as if it were an "existence" attribute. For example, OIL-SPILLS-INVOLVED-IN is "subvalue of INCIDENTS-INVOLVED-IN where is in OIL-SPILLS". The OIL-SPILLS-INVOLVED-IN flag is treated as if it were "if in class OIL-SPILLS". The only difference is that if the flag is not in the same record as the derived attribute, the system checks whether a set connects the two records. IF so, the appropriate RESULT clause is used.

The value of an "ordering" attribute, such as ORDERFORSHIP which is defined as "order by decreasing DATE within

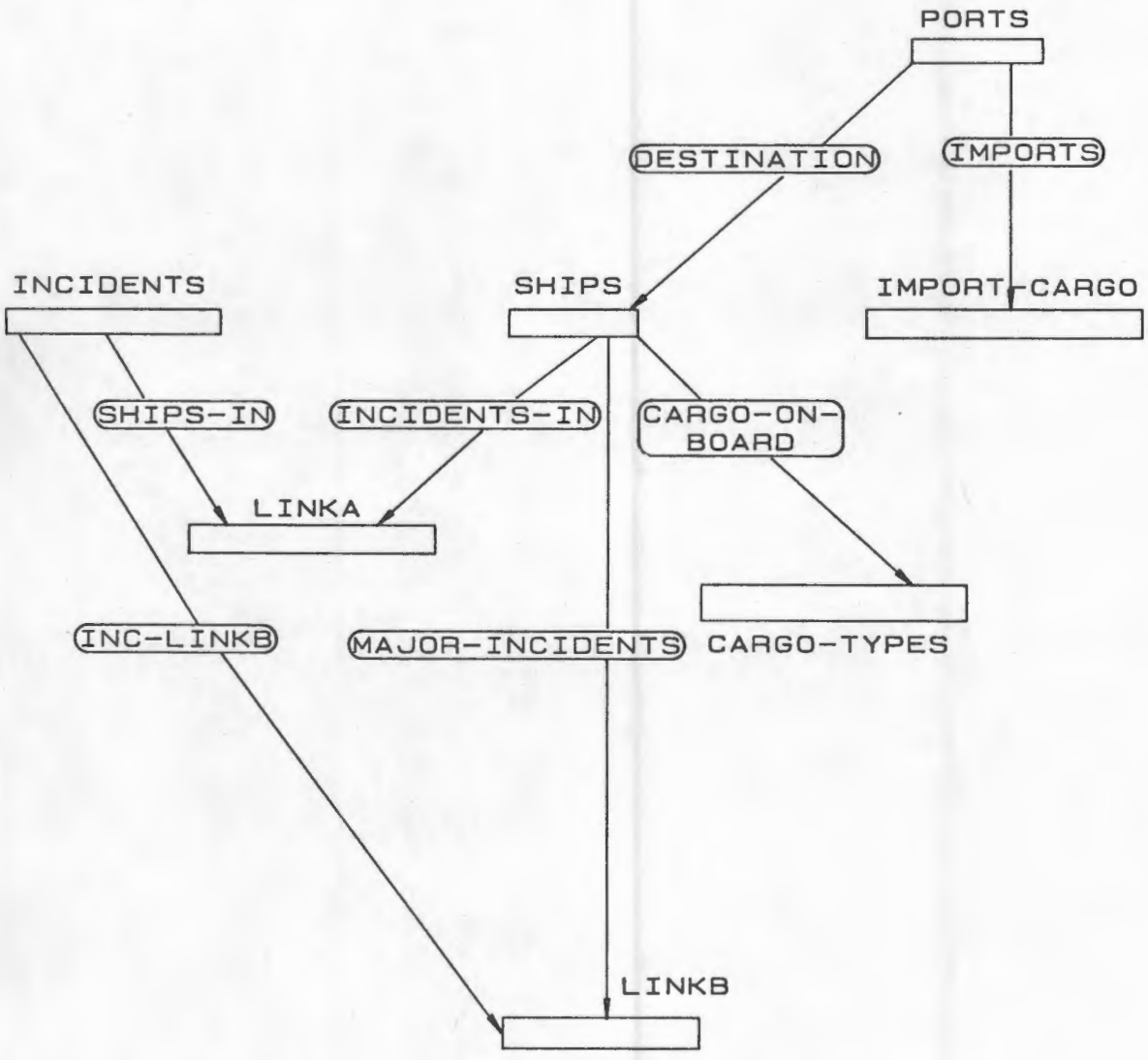


Figure 7.10 a) An example showing how "subvalue" derivations are handled.
This shows the network before dealing with "subvalue" derivations.
Two subvalue attributes exist here,
MAJOR-INCIDENTS is subvalue of INCIDENTS-IN; and
CARGO-ON-BOARD is subvalue of DESTINATION.IMPORTS

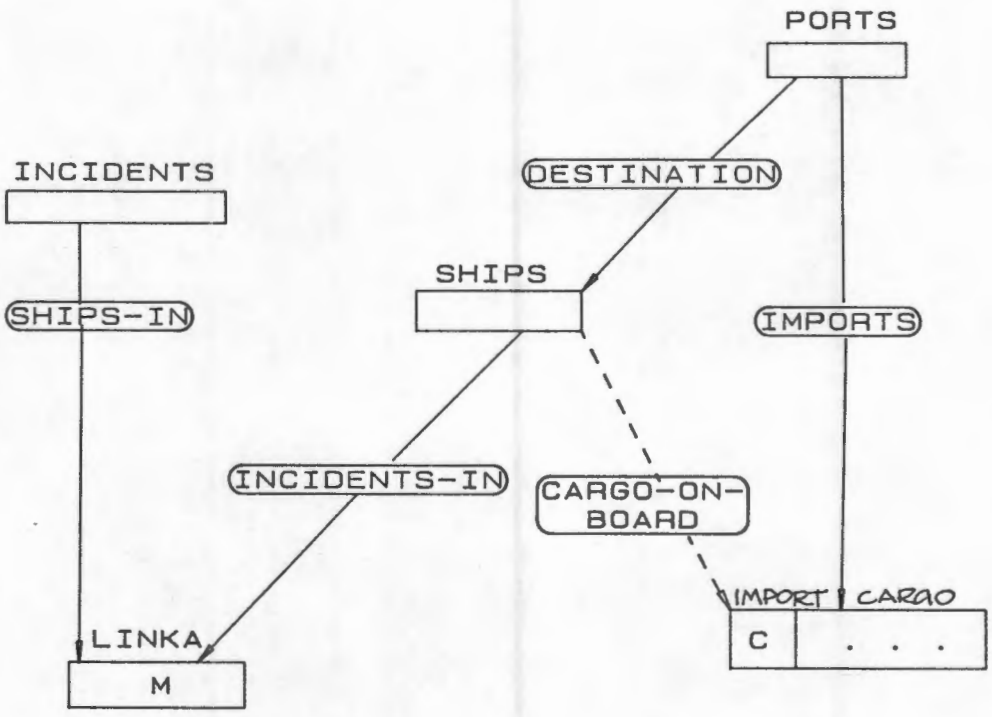


Figure 7.10 b) The example of fig. 7.10a, after the subvalue derivations have been treated. (M and C are the flags representing subvalues.) The broken arrow indicates manual membership.

SHIP-INSPECTED", cannot be verified using DMS 1100 integrity constraints. Thus instead of using a field for this attribute, the database set(s) involved are sorted (see fig. 7.11). This is done because the ordering is clearly significant; and the Nth value can be located using a "FIND (N, ...)" command [Sperry 1972]. To see how this is done, consider firstly an "ordering" attribute in class C, having no within clause. the set QENTRY -> C is created (if not already existing) and sorted according to the stipulated key(s). (In fig. 7.11, QENTRY -> INSPECTIONS would be sorted on SEVERITY). If any of the given keys are not in the C record, they do not contribute to the sorting. The ordering is not abandoned however, as a sort on the other keys will still be of benefit in the database. If a "within" clause appears, then instead of sorting set QENTRY -> C, the sets indicated hereby are sorted. For example an ordering "within SHIP-INSPECTED, COUNTRY-OF-REGISTRY" will cause the COUNTRY -> SHIPS set to be sorted on the key for SHIPS, and then SHIPS->C will be sorted on the specified "ordering" items.

Recursion: Since such an attribute is always implemented as a many-to-many mapping from class C to the same class C, it is always implemented as two sets C -> LINK, as suggested by Date[Date 1981]. The attribute from which it is derived becomes a flag in LINK. Thus if SUPERIORS is "all levels of COMMANDER", COMMANDER Becomes a "COMMAN" flag in the LINK record. The "SUPPRS" set is sorted on this flag, so that the COMMANDER is always the first member. It should be noted that if a recursive attribute has an inverse, the second C -> LINK set represents the inverse attribute. If SUBORDINATES is the inverse of SUPERIORS, then we have set SUPRRS from OFFICERS -> LINK and set SBRDNT from OFFICERS -> LINK, with COMMAN the only field of LINK. If the

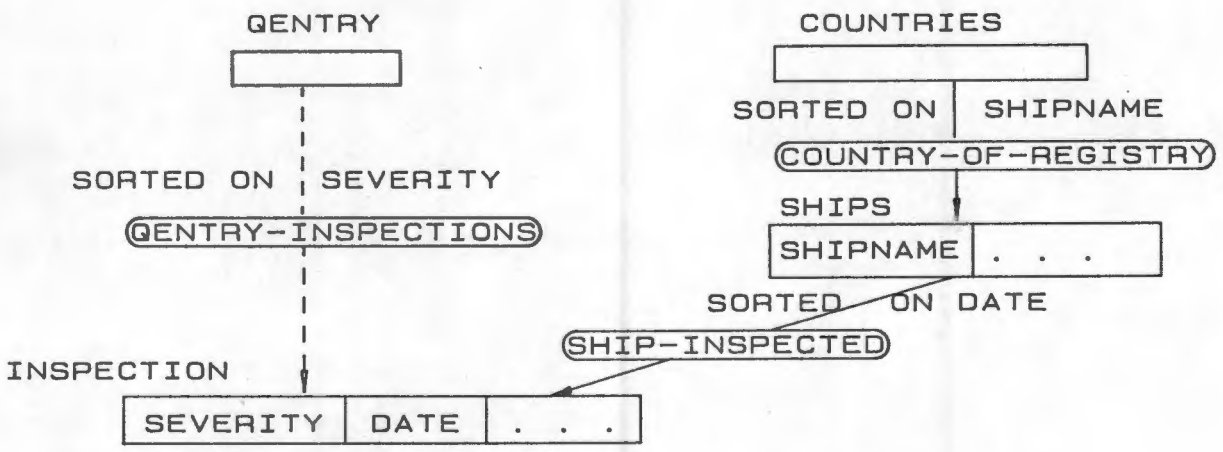


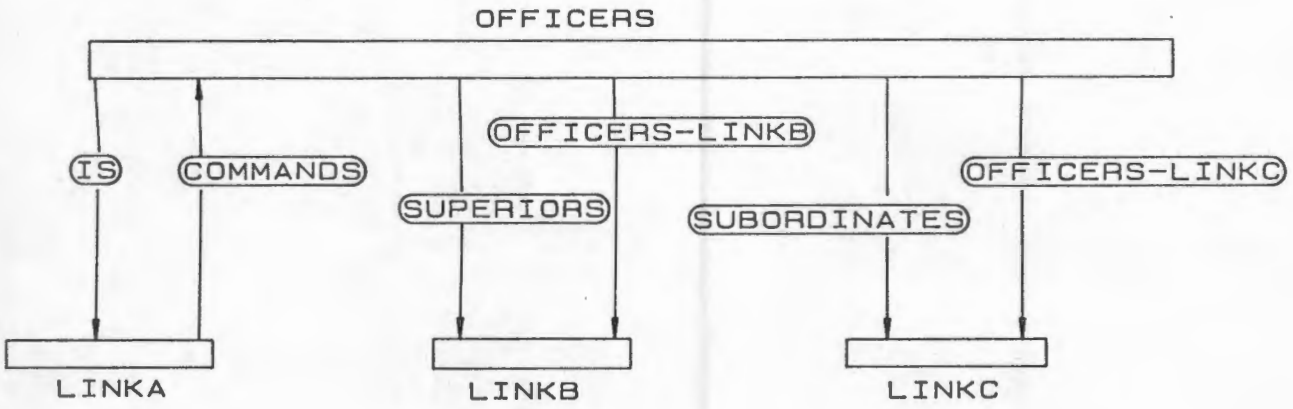
Figure 7.11 Examples of how "ordering" derivations are treated. Here INSPECTION-ORDER is order by decreasing SEVERITY;
and ORDERFORSHIP is order by decreasing DATE within COUNTRY-OF-REGISTRY, SHIP-INSPECTED.
(The broken arrow indicates a set added for the ordering.)

recursion and inverse relationships had not been specified, six sets and three LINK records would have been necessary. This is shown in fig. 7.12.

The item representing an existence attribute becomes "RESULT PROCEDURE QEQUAL ON THIS RECORD" using the relevant subclass flag, if this is in the same record. Procedure QEQUAL simply sets the one flag equal to the other. This would occur if say OILTANKERS had attribute IS-TANKER-BANNED defined as "if in class BANNED-OILTANKERS" and the BANNED-OILTANKERS flag were in the OILTANKERS record. A similar clause is used if the BANNED-OILTANKERS flag is in a record connected to OILTANKERS as member of some set. Otherwise, the RESULT clause cannot be applied.

RESULT clauses are attached to attributes having an "occurs" or "function" derivation, so that their value can be set by DMS 1100. A set is created to connect the record in which an "occurs" attribute appears to the record where the related item is, unless such a set already exists. The attribute becomes "RESULT COUNT ALL MEMBERS" of that set.
The situation is similar for attributes defined as a "function" of some mapping, but with "RESULT PROCEDURE P ON MEMBERS" of that set, P determined by the specified function.

If A is the "same as" B, and both are data items in the network, then it must be determined which is "above" the other. This is done by inspecting the first attribute in the mapping. If it is single-valued the attribute is "above" A. For example, if an OILTANKER's DATE-LAST-EXAMINED is the same as the INSPECTION-DATE of its LAST-INSPECTION and OILTANKER is above INSPECTION, then set OILTANKER -> INSPECTION is used to stipulate that



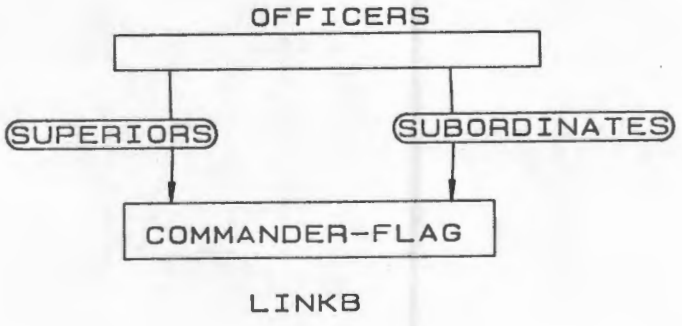


Figure 7.12 a) An example of how a "recursive" attribute, with an inverse, is implemented by ADD. The upper diagram shows what the network would be like if the recursion and inverse were not specified. The lower diagram show the structure obtained after handling the recursion derivation.

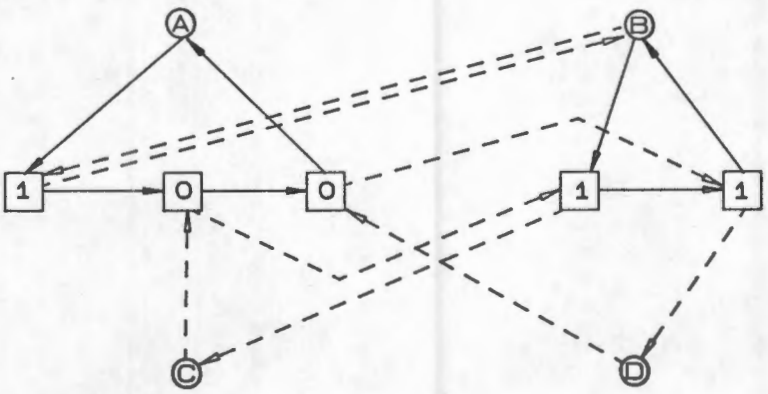


Figure 7.12 b) An example of a recursive attribute.
Here A commands B, who commands C and D.

DATE-LAST-EXAMINED is "RESULT OF PROCEDURE QEQUAL" using INSPECTION-DATE. If INSPECTION is above, then a set INSPECTION -> OILTANKER is used and a similar result clause appended to INSPECTION-DATE. In either case, the RESULT clause is specified only if a set already connects the two records. Of course If A and B happen to be in the same record, then "RESULT OF PROCEDURE QSAME ON THIS RECORD USING" B is defined for A. If either is implemented as set(s), not as an item, then it is removed from the database as it is redundant. (Example: if CREW is SAME AS STAFF, only one set SHIPS -> OFFICERS is necessary).

"Arithmetic expression" derivations are treated as follows. If A = f(B1, B2,...,Bn), and all Bi are in the same record as A, then A is made "RESULT OF PROCEDURE P ON THIS RECORD USING B1, ..., Bn". To generate procedure P, an entry describing P is made in file DBPFILE (as for subclass expressions). As the expression is processed, each term is written to this file, specifying the fields and records involved. A unique name for P is also determined.

Unless all the items are in the same record, the RESULT clause cannot be used and hence the value of A cannot be derived by the DBMS. This is because A must have a value class of real or integer and hence must be a field in some record R. The same applies to an attribute used in the expression. If it is in a different record, say R2, it must be above A in the network, as it can associate only one value with A, for the expression to be valid. Hence the Bi attributes cannot be connected to R as member of some set.

The remaining derivation is the Class derivation. This can state that a class attribute equals the number of members in the class,

or minimum/maximum/average/sum of an attribute taken over all members of the class. As the network has a QENTRY record at the top (as entry point) in which all class attributes are stored, a set linking QENTRY to the class concerned is added (unless it already exists). In this way the class attribute is connected to the record type it characterises. The appropriate RESULT clause is attached to the attribute.

7.5.2. Inter-Dependent Derivations.

The above schema modifications are not performed for attributes that are derived from the "CONTENTS" of some class. If the derivative is "same as contents" that attribute is dropped as it corresponds to members of the set for that class. For example CONSTITUENTS of CONVOY, "same as contents", is a redundant attribute. In other situations, a different derivation is indicated and the attribute handled accordingly. To illustrate this, consider firstly a "wherein" derivation. This reduces to "SAME AS" for an intersection or union involving "CONTENTS". This also happens with set difference where the second attribute is derived from "CONTENTS".

For set difference where the first attribute is derived from "CONTENTS", complementation is needed instead. This is implemented by means of a "RESULT OF PROCEDURE QNOT" clause. This procedure sets one flag equal to the negation of the other. If both attributes in a wherein derivation are "CONTENTS" attributes, the derived attribute is erased being equivalent to "same as CONTENTS".

If an attribute A is the subvalue of some other attribute that was

indirectly derived from a "contents" attribute, then A is implemented as a flag, not as sets. For example, if attribute CONSTITUENTS of CONVOY has derivation "contents" and OILTANKER-CONSTITUENTS is "subvalue of CONSTITUENTS where..." then OILTANKER-CONSTITUENTS becomes a flag in member record of the set representing the CONSTITUENTS.

The other derivations either cannot be applied to "CONTENTS" or are trivial to implement (e.g. NUMBER-OF-SHIPS-OF-THIS-TYPE equals the COUNT of members in a SHIPTYPES -> SHIPS set). Since "ordering" attributes are removed from the database, all their derivatives are also abandoned or modified in a similar way to CONTENTS derivations.

The discussion in the preceeding section ignored the fact that derivations are inter-dependent. That is, if A is derived from B, B itself can be a derived attribute. When considering this problem it was realised that three kinds of action are taken to handle derivations:

I - sets are added to the database

II - attribute implementations are eliminated

III - set members can be changed (for "wherein", "recursion" or "subvalue").

Thus, all type II derivations are handled first (that is, "contents" and "ordering" derivations) and each time such an attribute is removed, all its derivatives are dropped or altered, as are all their derivatives, etc. Thereafter, the type III derivations are done. Since these can be inter-dependent, any items in the old member are first transferred to the new. Type I derivations are handled last, once all erasing and altering of sets is complete. This is because the existence or non-existence

of certain sets is central to the processing of such derivations ("occurs", "function", etc.). This method was used in preference to recursive handling of derivations, because the SDM specification resides on (Sequential) Pascal files. In this way, they are RESET far less often than would be the case with a recursive program.

7.6. Physical properties.

Simplicity was the primary aim when appending physical properties to The logical schema. thus all records were given a VIA SET location mode, except for the QENTRY record at the 'top' of the network, which is 'direct'. Records not participating as automatic member of any set, become the automatic member of a new set with QENTRY as owner. In this way, all records except QENTRY have a 'parent' to connect them to the rest of the network. If such a set QENTRY->R already exists, then it is chosen as R's VIA set. As there is only one such set occurrence in the database an R occurrence can then be accessed immediately, instead of needing to trace a path through the network. Otherwise the first set of which R is an automatic member becomes its "VIA" set. This corresponds to the first such relationship involving R that was given in the SDM specification; this is usually the most significant one. In general, it cannot be assumed that the ADD system will choose the best 'via' set for all records. To do this, it would need a great deal of extra information from the user as to the relative importance of relationships (which user may or may not be able to supply). hence it was decided to spare the user the decision, and leave it up to the database administrator to make any 'via set' alterations he sees fit.

All sets for which 'order sorted' has not already been specified, are ordered "last". This is the fastest when setting up such a set and the most efficient when processing it [Sperry 1972]. Furthermore, with sets having only one member occurrence (such as a superclass-subclass set), this allows a 'find member' to be issued without the need for specifying items with which to identify the member. Sets that are sorted are those which the user required (by means of an 'ordering' attribute) and those used in ALIAS clauses, together with those sorted on some indicator-field. For example, INCIDENTS is sorted on OILSPILLS flag. All sets are assigned a mode of chain, and prior links are not specified, to save space and time. (Prior links will save time only in special circumstances [Sperry 1972]). Pointer arrays and indexed pointer arrays are not used because this is only worthwhile if the member record is a member of more than one set and processing requires frequent use of more than one of these sets in one query.

Set selection is 'current of set' wherever this is possible. The exception occurs when two or more sets have identical owner and automatic member record types. This would occur for example with two sets both having owner VOYAGE and member OFFICERS, such as CURRENT-VOYAGE and LAST-VOYAGE. In such situations two owner occurrences of the same record type must be identified when a member OFFICERS is stored. This problem is overcome in DMS 1100 by using an ALIAS for the identifier of the owner record, on one of the two SET SELECTION clauses. (eg. CURRENT-VOYAGE uses VNUM and LAST-VOYAGE uses VNUM-ALIAS). Such situations are recognised by the ADD system, and dealt with accordingly.

The solution just outlined cannot however be applied if the owner

does not include a complete key. For example if the key of a VOYAGE is START-TIME, SHIP; and the latter is a set SHIPS -> VOYAGE. Then a multi-level path is required to identify any record below VOYAGE. So aliases are provided for start-time (in VOYAGE) and name (in SHIPS). This is illustrated in fig. 7.13. Such a situation also arises with subclasses, where the key (in the base class) can be any number of records above the subclass record. Each subclass thus needs an alias for the key of its superclass and 'using' keys for intermediate records. A recursive algorithm was designed for tracing a path from such records 'up' to a record with its key completely represented by fields in that record. This also assigns ALIAS or USING items along the path.

Where the key of a record R1 comprises more than one attribute which is not a field of R1, aliases are insufficient to enable two such records to be identified at one time. An example is given in fig. 7.14. In that example, class VOYAGES has key <START-PORT, ROUTE>, both attributes represented by sets. Problems arise if two sets VOYAGES->OFFICERS exist. These are solved as explained below. In general, if R1 is such a record and two or more sets R1 -> Ri exist, dummy 'LINK' records are created for all but one of these sets. Each of these sets Si become two sets R1-LINKi and LINKi-R2, both with SET SELECTION CURRENT OF SET. At first this may appear to waste a lot of space, but in fact only three extra words are taken for each set split in this way. To store an OFFICER in our example, requires the following:-

1) find VOYAGES owner for the last voyage
2) find first LINK within VOYAGES-LINK-SET.
3) find VOYAGES owner for CURRENT-VOYAGE
4) store OFFICERS record.

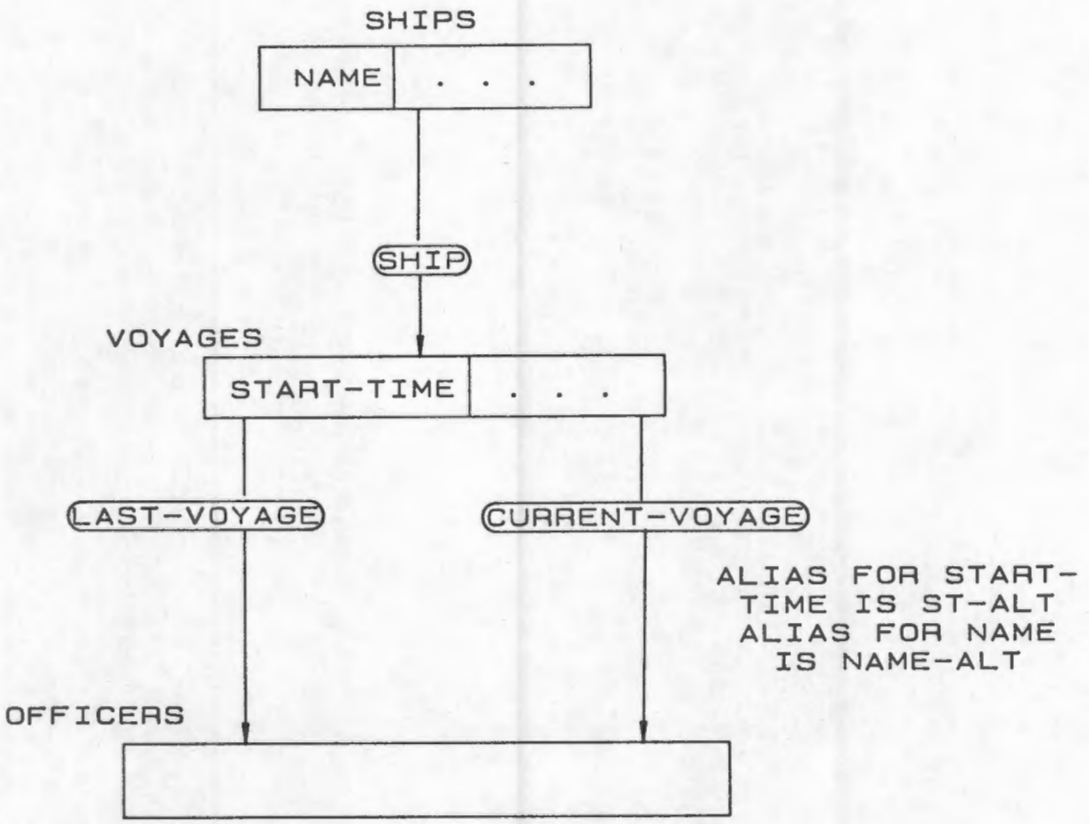


Figure 7.13 An example of where an ALIAS clause is needed.

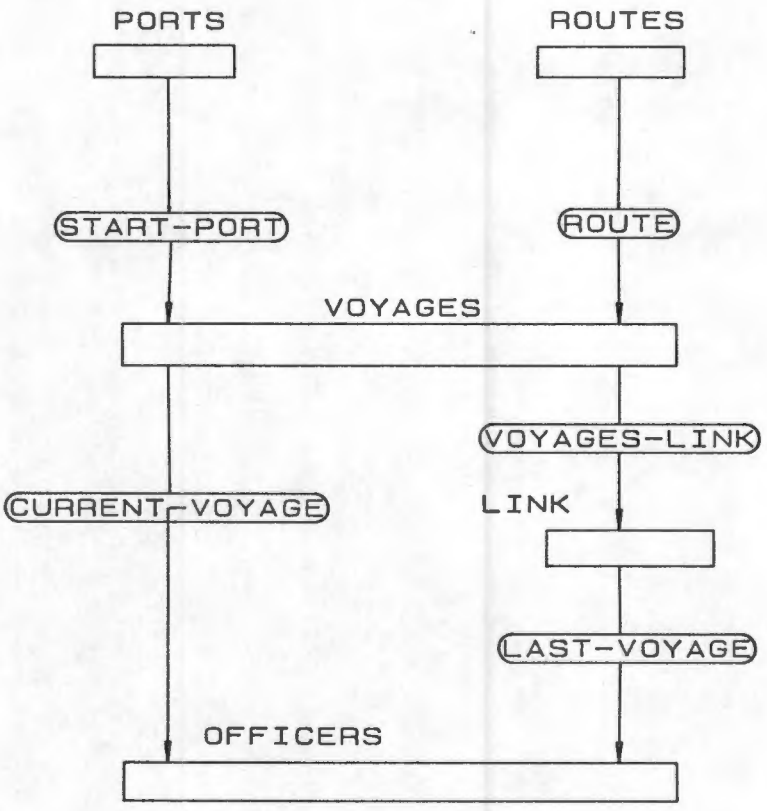


Figure 7.14 An example where an ALIAS clause cannot be used, showing how this situation is catered for.

It should be noted that this type of situation occurs but rarely in most databases, as the problem only arises with automatic members, and usually when the special circumstance of having two sets R1-R2 does arise, one of these is a manual relationship. In this example LAST-VOYAGE should really be a manual set, hence no problem would occur. Another important point is that the choice of which set to 'split in two' is not made arbitrarily. If one of the sets is used in a RESULT clause, then the other set(s) are split; and RESULT clauses involving a database procedure are given preference, as the others are simpler to change to reference the new set (R1-LINK). In conlusion therefore, a simple set of physical parameters augments the logical structure. The only complexity arises with sets for which a multi-level path must be defined, with USING or ALIAS items for intermediate records.

[Date 1981] Date, C.J. "An Introduction to Database Systems", (3rd ed.), Addison-Wesley, 1981.

[Sperry 1972] Sperry Univac, "Database Management System (DMS 1100)", UP 7909, Rev. 3A, St. Paul Minn., 1972.

[Taylor 1976] Taylor, R.W. and Frank, R.L. "Codasyl Database Management Systems", ACM Computing Surveys, Vol. 8, No. 1, March 1976, pp. 67-104.

8. CHAPTER 8. TUNING FOR PERFORMANCE.

8.1. Introduction.

The ADD system generates a conceptual schema and incorporates simple physical specifications to create a complete DMS 1100 schema, which can be used as a prototype design. This schema may have to be tuned to improve its efficiency. As this depends on knowledge of the data itself - that is the relative importance, volume and frequency of use of each record and item, this is a process which is best done manually. The following paragraphs provide guidelines for manual alteration of the schema to meet performance requirements.

8.2. General Guidelines for Improving Efficiency.

In improving the efficiency of a database, the general method is to identify overhead situations and then to resolve this by altering either the physical properties or the logical structure at that point. To alter the database in this way, it is necessary to gather a substantial amount of extra information from the user. For each record type, the expected: number of occurences, frequency and type of access, and selection criteria must be established. Where the frequency or type of access for an item differs from that of the record of which it is a part, the designer must be made aware of this. Further, the frequency and type of use of all relationships and the expected number of members of all set types should be supplied by the user. The ease

and accuracy with which the user can estimate this information depends on the type of data in the database. In some situations he may even be able to provide additional data such as common patterns of access to groups of items and/or records, security needs, time constraints on availability of data, long term archiving and any high-priority programs requiring special response times [Sperry 1977]. For the most part however, the user will have difficulty telling even the basic requirements, especially if inexperienced or new to computerisation.

Once this information is available, potential overhead situations can be located and rectified. The most common overheads are listed below.

1) maintenance of sorted sets. Especially when sets are large or volatile, order LAST or NEXT is preferable.

2) poor choice of the "VIA" set for a record type. If this set type is not the most frequently used relationship when accessing such records, or if it has very many members, an alternative should be used.

3) extremely large records. This results in few records per page and hence more page changes (I/O's) than would be necessary with smaller records. This can be rectified by having larger pages (which has its disadvantages) or, rather, by splitting off rarely-used items of such records, eg. ADDRESS, to form a separate record type. large items whose actual length varies from one record occurrence to another (e.g. a text) should be reduced to the smallest reasonable size, and an "itemextension" member record used for larger items.

4) implicit many-to-many relationships. As an example, suppose most INCIDENTS involve only one SHIP, but a few may involve more than one. If a set INCIDENTS -> SHIPS is used there will be

duplicate occurrences of some SHIPS. This causes serious overheads if the SHIPS record is large or is involved in many sets. The set should be replaced by an explicit many-to-many relationship.

5) page "thrashing" in large set occurrences. Traversing the logical ring pointers may involve several page-swops depending on how much the physical ordering of members differs from their logical order. An ORDER of LAST minimises this possibility (and hence is used as a default in ADD). The use of PRIOR and/or OWNER links alleviates this, as the DBMS will then use the shortest path to a record. If the members are small and volatile, they can be combined into fewer record occurrences. Alternatively the member record can be replaced by a repeating group in the owner, with a "memberextension" record type for instances where the size of the repeating group is inadequate. In this way, fewer DML commands will be needed to process the set, which is a considerable saving because of the "base" overhead involved in any execution of a DML command.

6) too many secondary sets. If a record is the member of more than one set, one is called the prime set and the others are termed secondary. If the member is accessed frequently or is highly volatile, the mode of these sets should be changed from CHAIN to POINTER ARRAY. This reduces the number of I/O's required when traversing each set of which it is a member.

7) incorrect choice of location mode can result in poor placement of records in a file or unacceptably slow accesses. Good discussions on choice of location mode can be found in [Ullmann 1980, Date 1981].

8) lack of redundancy. It is sometimes advisable to introduce a certain amount of so-called "controlled redundancy" to facilitate

database processing. Factors such as stability and access frequencies of the items concerned must be considered as there is a trade-off here between saving time and space.

9) unnecessary record types. This can occur in certain situations where two or more virtually identical record types have been specified, such as INCIDENTS and OILSPILLS. Whether this overhead is justified, or whether the record types should be consolidated, depends entirly on the meaning and use of the data.

10) proliferation of outdated information. It is often advisable to replace a large amount of historical data by a few items which summarise that information, such as averages and totals.

8.3. Modifying a Schema designed by ADD

There are essentially two ways in which an ADD schema can be tuned, viz. the physical criteria can be altered; and/or the logical structure can be adjusted. These will be discussed separately in what follows.

8.3.1. Physical Considerations.

The major change that will probably be required is that of examining the location mode of (major) record types. Some records may be justifiably more suited to DIRECT, CALC or INDEX SEQUENTIAL placement; while others may need the choice or their "VIA set" altered so that such records are found using the best path. Another probable alteration is the use of more than one file for the data, and allocating records to these files according to privacy requirements and/or expected patterns of access. This

could be automated, but would require a fairly sophisticated algorithm to assign records to files suitably, as well as an intelligent user interface to be able to extract the relevant facts about the data for use by this algorithm. This become more complex if pages within files and/or INTERVAL clauses are used; hence this process is probably best done manually by the DBA. Extra pointers can be added where appropriate for faster access to records. In this way LINKs TO OWNER and PRIOR LINKs could be added to important (frequently-used) sets. Where a record is member of several sets, the mode of these sets could be changed from "chain" to "pointer array", if fields in more than one owner will often be needed at the same time.

Other alterations which can be made at the discretion of the DBA, but which have a lesser effect, are: encoding records and/or items; changing the set order of some sets or their set occurence selection; specifying "LOOKS", for recovery; choosing other area size(s) with different pages designated for overflow or a different load factor; addition of security locks on areas, records and/or items [Sperry 1972]. It should be noted that the ADD program which creates a schema source listing according to the QDB-database can handle any DDL clause. Thus even clauses not used currently by the ADD system (e.g. ENCODING and ACCESS LOCKS) can be included in this database and the corresponding schema listing will still be correctly generated.

8.3.2. Logical Design Modifications.

The logical view of the enterprise should only be altered if changes to physical criteria are insufficient to meet performance

requirements. When changing the logical design, the outputs provided by the ADD modules should be examined to see why the particular structure arose and all places where it is used (such as in RESULT clauses). There are several logical database modifications which can be made; which are worthwhile depends entirely on the meaning, usage and volume of the items concerned. These modifications are:

1. Where a special record and set type have been created for a subclass, these can be replaced by extra fields in the owner (superclass) record. These fields will simply be blank for all objects not in the subclass. This may be useful if the subclass forms a substantial portion of the superclass, or if the extra fields require relatively little extra space (few small fields or few subclass occurrences).

2. A multi-valued attribute may be better suited to being a repeating group than a separate record type. The situations where this is advisable have already been discussed.

3. Derived items can be removed from the schema, along with the corresponding RESULT clause(s). Factors to use in deciding which sets and items to remove, are: importance, frequency of use and complexity of the RESULT clause (e.g. did it require a RESULT PROCEDURE, which involves large overheads?).

4. Large records can be split or small records consolidated, if appropriate.

5. Several sets with the same owner record type can be consolidated into one set type with different member types, depending on how the members will be used. This can make for more efficient access to member occurrences [Teorey 1979]. Thus for example sets SHIPS->PORTS and SHIPS->COUNTRIES can be combined into one set type PLACES-VISITED with owner SHIPS and members

PORTS and COUNTRIES. In conclusion it should be stressed that altering the physical parameters as suggested should be sufficient to make the prototype perform acceptably. Adjustments to the logical structure will probably not be required.

[Date 1981] Date, C.J. "An Introduction to Database Systems", (3rd ed.), Addison-Wesley, 1981.

[Sperry 1972] Sperry Univac, "Data Management System (DMS 1100)", UP 7909, Rev. 3A, St. Paul Minn., 1972.

[Sperry 1977] Sperry Univac, "Database Design Instructor Guide", UE-786,Sperry Univac Education Centre, St. Paul Minn., March 1977.

[Teorey 1979] Teorey, T.J. and Fry, J.P. "The Logical Record Access Approach to Database Design", Proc. of Int. Conf. on the Entity-Relationship Approach to Systems Analysis and Design, Los Angeles CA, December 1979.

[Ullman 1980] Ullman, J.D. "Principles of Database Systems", Computer Science Press, Potomac MD, 1980.

9. CHAPTER 9. SDM CRITIQUE.

In order to evaluate SDM, we need to firstly determine criteria for 'good' data models. Of those mentioned in chapter 3 the most important are ease of use - the ability to obtain a correct and complete specification from an unskilled person; semantic expressiveness - how much information is contained in a model; and compatibility with conventional database systems - how much of the model can be implemented. SDM was chosen as a basis for ADD because of its success in achieving the first two of these objectives. The construction of the system has however revealed that its variance with conventional DBMS's detracts from its overall value. Before the model is evaluated, The good and then the weaker features of SDM will be discussed.

9.1. Valuable SDM Concepts.

The most remarkable quality of SDM is that it is at the same time simple and yet abounding in special modelling features. A data model should primarily be easy to use and hence to conceptualise. A clear picture of an SDM model as a collection of classes having attributes to describe them, should present a novice with no problems: there are only two basic modelling elements. If the idea of subclasses, groupings and attribute derivations are introduced to him gradually, as in the ADD system, his view of the model should remain clear and workable. He can present it directly without needing to rely on a diagrammatic representation (which could become unmanageable as the specification grew) or on complex predicate calculus expressions.

This clarity and simplicity is enhanced by the concept of classes representing themselves and not being referenced by keys, as in the Relational Model and others. This leads to an exceptionally neat method of presenting an association between two classes A and B. When describing say A, it is natural to want to include its relationship to B. This can be done by making this association an attribute of A whose valueclass is B. Thus specifying the INCIDENTS properties DATE (type DATEs) and INVOLVED-CAPTAIN (type OFFICERS) is equally easy, without the complexity of distinguishing the former as an 'attribute' and the latter as a 'relationship'. This is a major advantage if SDM is to be used by the computer-naive, who are not accustomed to thinking in terms of 'entities', 'attributes' and 'Relationships'. If involved-captain had to be given a valueclass OFFICER#, or Surname+Initials, this would greatly aggravate such a user's task; as it is well known that the idea of a key is so foreign as to be one of the most difficult for users to grasp [Martin 1973].

Of course the above is not the only manner for specifying relationships: inverses and matching (where an association can be represented as a class with its own attributes) are alternative methods. Indeed, the principle of relativism is central to the design of SDM - allowing for as much redundancy in the model as possible. This alleviates the mammoth but important problem of obtaining a 'Complete' specification, as well as providing a mechanism for detecting inconsistencies.

There are also many semantically expressive constructs in SDM. For example, the concept of grouping classes seems such an obvious one that it is suprising that this has not been incorporated into other models. Providing three types of grouping class definition

forces the user to think clearly and stipulate only meaningful groupings. This is also true of IS-A hierarchies in SDM, where generalisations must be defined by supplying the precise subclass interconnections. Generalisation can be applied to relationships, if these are represented as match classes.

The richness of the model is most evident when one considers its simplest construct, the attribute. The specification that duplicates, nulls and shared values are not allowed for particular attributes can be incorporated into a database to enhance its integrity. Class attributes cater for directly including in a model relevant properties of the organisation as a whole. The inclusion of derived data enhances the model's usability and naturalness. Shipman has said that SDM "does recognise the potential inherent in the notion of derived data. In addition, this work includes a great deal of pioneering research into the descriptive capabilities needed to represent, in the database, useful semantic properties of the real world being modelled" [Shipman 1981]. It can be seen that the diversity of useful modelling tools together with the principle of relativism make SDM one of the most semantically expressive models; and that it is at the same time simple and easy to use.

9.2. Problems and Shortcomings.

The most unsatisfactory aspect of SDM is that some of its features cannot be effectively implemented using a conventional DBMS. Having thoroughly investigated the conversion of such a model to a Codasyl schema, several instances were discovered where incorporating or verifying the integrity of SDM features is either

impossible or inefficient. It becomes apparent when studying these areas of difficulty that all these problems would also exist with hierarchical systems, and some with relational databases too.

Most of the difficulties arise because of the flexibility afforded the data model designer in the use of mappings. This enables two values which are very indirectly associated to be stipulated as related in some way (in an attribute derivation or interclass connection). In a CODASYL database, where the corresponding items would be in different records far apart in the network, this relationship cannot be verified and thus an inconsistency could arise. The 'semantic distance' of two attributes is a measure of how closely related they are. This should be used to impose restrictions on mappings. They should be limited to the point where they are verifiable by the DBMS. Otherwise, where one attribute is derived from a lengthy mapping, or where an interclass connection is defined on an attribute that is not directly part of its base class, a choice must be made. Either the attribute or class must be removed or the integrity of the database is compromised. The same can be said of "class derivations", where the attribute involved is not directly part of that class.

There are other instances too where the absence of rules on the application of a modelling tool can result in incompatibilities with a Codasyl structure. For example, if a grouping class is defined on "common value" of several attributes, of which more than one is multi-valued, the grouping cannot be checked. The same applies if more than one of these attributes represents an entity type (class). Similar difficulties occur if one of these attributes is multi-valued and another is a class. The grouping

of members in an enumerated grouping class also cannot be verified, as was explained in chapter seven.

Some of the attribute derivations allow the specification of qualities which are not supported by the integrity constraint mechanisms of a Codasyl DBMS. For example recursion cannot be completely checked. If SUPERIORS is defined as all levels of Commander, then if A1 commands A2 who commands A3, we can ensure that A1 is in A2's set of SUPERIORS, that A2 is in A3's set of SUPERIORS, etc. But we cannot ensure that no extraneous members become SUPERIORS. The idea of being able to specify this recursion "up to <N> levels" is even further removed from the capabilities of a Codasyl (if not any) DBMS.

The value of an ordering attribute cannot be verified either, the best that can be accomplished in a network database is to sort the corresponding sets on the specified attribute(s). The presence of an explicit ordering attribute should not be necessary for the inclusion of an ordering facility. Furthermore the "within" section is rather confusing; it is not easy for a user to employ this to correctly stipulate where ordering is important.

The "existence" derivation seems to be of limited use in a database (example : IS-TANKER-BANNED). This information should be easy to obtain without the need for storing an extra flag in the record. The only real advantage of such a derivation is when the derived attribute is itself to be used in a derivation.

A subvalue or subclass definition which involves testing the membership of one set against another set, e.g. whether HOME-COUNTRY-PORTS is contained in PORTS-OF-CALL, also cannot be handled using Codasyl CHECK or RESULT facilities. Whereas

specifying that some attribute equals the number of values of another attribute (eg. NUMBER-OF-INSPECTIONS) is useful and implementable, the stipulation to ignore duplicates in this counting is not.

SDM has been criticised for not "expressing its ideas within a concise notational framework" [Shipman 1981]. This is evident when one contrasts an SDM specification, such as that in appendix D, with a Daplex one, such as that given in chapter three (fig. 3.7). This would present a problem if an adequate user interface such as that of the ADD system, did not exist. It is a rather serious disadvantage however if data models are considered as a communication medium between designer and user.

Besides the abovementioned problems a few omissions are also apparent. One of the most important is the fact that 'inverses' are optional. This was done to prevent any unimportant or insignificant attributes from appearing in a specification. Thus attribute INCIDENTS-INVOLVED-IN of class SHIPS may be given, but class INCIDENTS may only have attributes DATE and INVOLVED-CAPTAIN. In this case, there is no way of telling whether the SHIPS:INCIDENTS relationship is 1:many or many:many. To preclude duplication of SHIPS records, the latter must be assumed. Clearly if inverses were obligatory and 1:many relationships used wherever necessary, a simpler, more effective network would result. SDM is probably best altered to say that inverses must be supplied to indicate many:many relationships (less common); that is, the default should be 1:many. A similar problem occurs with user-controllable grouping classes, where it is not known whether each object will belong to only one group or to several (example : convoys). There is also no facility for

specifying the PICTURE clause of a real value (e.g. it is not possible to state that name class RANDS has 2 decimal places); nor to stipulate that an item occurs exactly M times.

The model does not include any dynamics constructs. However as has been explained, this was considered an advantage rather than a shortcoming. What may be viewed by some people as the most serious omission however is the fact that no allowance has been made for specifying functional or multivalued dependencies. This will be discussed in section 9.4, after a modified SDM is proposed.

9.3. Suggested Modifications to SDM.

The most important adjustment required is to limit the applications of mappings. No more than two attributes should be allowed (i.e. only mappings of the form A.B should be permitted). This ensures the related attributes will be connected by a set in a Codasyl schema. Class attributes should not involve mappings, but should only involve attributes that are properties of the class under consideration. Similarly, subclasses of class C and grouping classes defined on C should only use attributes of C in their definition.

In addition to this, the number of attributes involved in an expression-defined grouping should be curtailed. Attributes of non-standard valueclass (i.e. which map to database entities, not strings) should not be used, nor should more than one multi-valued attribute. The former is unnecessary, as the groupings will already be represented by sets in the database. The latter cannot be verified, and would probably be more useful and meaningful if

separate grouping classes were specified instead. As an example, SHIPGROUPS, a grouping "on common value of PORTS-OF-CALL, CARGOTYPES", is probably better implemented as CARGO-GROUPS and PORTOFCALL-GROUPS.

Some attribute derivations should also be altered. For example, the ordering of members should be specifiable without the need for an attribute to explicitly represent this. Recursion should always imply to all levels, as "to N levels" is unrealistic in the context of existing database management systems. The same applies to the stipulation that only unique members be counted in an "occurs" derivation. Inverses should be specified wherever these are 'to one'. That is, the default should be 'to many', as advocated by Hubbard [Hubbard 1979]. The same should be done with user-controllable grouping classes, so that there are no amiguities in the model - as is the case at present. Formats of items in terms of number of characters, number of decimal places etc., should be part of the model. Other derivations could also be considered for inclusion, such as Cartesian product.

9.4. Evaluation of SDM.

It is evident that there are some contentious issues surrounding SDM and its general suitability to database modelling. One of these is the absence of functional and multivalued dependencies (FD's and MVD's). The important point in this regard is that these concepts are subtle, and not easily learnt or understood even by people in the computing profession. Thus whereas a systems analyst or DBA would be able to specify them correctly if sufficiently familiar with the enterprise, it is unrealistic to

presume that users could do so. Indeed, in the process of testing the ADD system with non-computer users, it was clearly established that they experienced much difficulty with the concept of functional dependencies and still greater trouble in recognising these in their environment. (Multi-valued dependencies were not even attempted!) Although an FD feature could have been included in SDM for the astute user who might be able to employ it intelligently, this would confuse the majority, upsetting the simplicity of the model and compromising the correctness of its specifications. Furthermore, it is still doubtful whether all FDs (or MVDs) would be recognised; hence the relations and records generated may still violate 3NF (or 4NF). Some FD's and MVD's can be deduced for a given model (as is done in ADD), but there can be no guarantee that all are recognised. It is best for a user to employ a simple model without these two concepts, possibly adding them afterwards under the guidance of the DBA or analyst.

Considering the wealth of modelling constructs available in SDM, the question arises as to whether there is any benefit in creating a highly semantic model when that on which it is to be implemented (network/relational/hierarchical) does not support all its semantic features. After all, a data model should not be confused with a data dictionary. The latter is intended to store as much information about the environment as possible for reference purposes; while a data model is intended as a basis for database design. One could contend that semantic models are necessary for the database management systems of the future. However, if at present it results in inefficient or inconsistent databases, its benefit is questionable. Semantic features are best included in a model, but with their usage limited to being within the scope of conventional DBMS capabilities.

The principle of relativism raises the issue of whether a straight forward model with few modelling tools is better than one which has a large, redundant set of modelling constructs. (that is, is a "fixed" or restricted interpretation better than a "free" interpretation?) SDM has a rich set of modelling primitives and allows a rather "free" interpretation. The important issue however is whether or not a model with many constructs is difficult to understand. That is, it is the existence of obscure or complex tools that is undesirable. In general, systems with many modelling tools tend to include some that are complicated and thus make for an overall complex picture. This is true of CSDL [Roussopoulos 1979] and Database Skeletons [Chang 1978] for example, but not of SDM, whose modelling elements are all simple to use and understand. Furthermore, the idea of providing several methods by which associations can be specified increases the probability of obtaining a complete model of an application environment.

SDM is not as suitable for relational schema design as for CODASYL, because it does no include specification of dependencies. The problem of greater semantic expressiveness in SDM than in the relational model further reduces its suitability for relational database design.

In conclusion therefore, it is felt that SDM is a 'good' model on which to base a network or hierarchical database design, but that some of its features should have restrictions placed on their usage, to enhance its compatability with conventional Database Management Systems.

[Chang 1978] Chang, S.K. and Cheng, W.H. "Database Skeleton and its Appications to Logical Database Synthesis", IEEE Trans. on Software Engineering, Vol. SE-4, January 1978, pp. 18-30.

[Hubbard 1979] Hubbard, G.U. "Computer-Assisted Logical Database Design", Computer-Aided Design, Vol. 11, No. 3, May 1979, pp. 169-179.

[Martin 1973] Martin, J. "Design of Man-Computer Dialogue", Prentice Hall, Englewood Cliffs New Jersey, 1973.

[Roussopoulos 1979] Roussopoulos, N. "Tools for Designing Conceptual Schemata of Databases", Computer-Aided Design Vol. 11, No. 3, May 1979, pp. 119-120.

[Shipman 1981] Shipman, D. "The Functional Data Model and the Data Language DAPLEX", ACM Trans. on Database Systems, Vol. 6 No. 1, March 1981, pp. 140-173.

10. CHAPTER 10. CONCLUSION.

In this final chapter, the ADD system is evaluated in terms of the design criteria stated at the beginning of this thesis. Possible changes and extensions to ADD are discussed and suggestions for future research are made. In conclusion, the design principles that have emerged from this study are presented.

10.1. Evaluation of ADD.

The primary objectives of the ADD project were to automate the process of requirements specification and to generate a DMS 1100 prototype schema. The requirements were to be formulated as an SDM model and as much as possible of the semantics of this model was to be incorporated in the database. Synthesis of a relation schema was also planned. A secondary objective was to analyse the potential of SDM as a basis for database design, in the light of experience gained in developing ADD. This last objective has already been discussed in the previous chapter and hence is not covered here. The others will now be considered in turn.

The first goal was to provide a system whereby computer-naive users could easily and correctly specify an SDM model. The user interface was accordingly designed for maximum simplicity, and the features of SDM were introduced gradually to avoid confusion. The only way to determine whether ADD meets the ease-of-use criterion is to observe how novices perform using this system. Such an experiment was undertaken. This showed that whereas users do experience some difficulty initially, if they are assisted by the database designer during their first few minutes on the system,

they proceed satisfactorily thereafter. The main reasons for this seem to be the user's awe of using a computer terminal, and his misinterpretation of terminology such as 'object' or 'identifier'. Once this has been clarified, the user can largely be left alone to complete the specification. Naturally some problems may still arise, but only in exceptional situations. When these test took place, a user manual had not been written for ADD. The problem areas highlighted by this experiment were used to write the User Manual for the system. Indeed, it is intended that this manual be updated periodically, as experience increases our understanding of users' difficulties. The usability of the present version of ADD can be summarised as follows: After reading a few pages of a User Manual to clarify the concepts on which ADD is based, or after a short discussion with the database designer, a computer novice of average intelligence should be able to create an SDM specification using ADD, with little (if any) further assistance.

Regarding the second objective, we see that the ADD system does indeed produce a prototype DMS 1100 schema. The quality of this prototype is of interest here. Its strength lies in its choice of a logical structure, which should require minimal if any alterations. The wealth of information in the SDM model has been used to define records, sets and items in the best possible way. The characteristics of attributes, interclass connections and attribute derivations which enhance one's understanding of the organisation, are utilised to improve the network. As the logical structure is based solely on a description of the data, and not on functional requirements, it should not need to be reorganised when processing requirements change. Integrity constraints are included where permitted by the DMS 1100 CHECK and RESULT clauses. As these facilities are more restrictive than the SDM modelling

constructs such as mappings,

the SDM constraints cannot always be incorporated. However, the prototype generated by ADD will definitely specify integrity constraints wherever this is possible.

A simple physical structure is used for the prototype, since optimising physical parameters was considered beyond the scope of this system. However, ADD does relieve the human designer of much of the tedium associated with the specification of physical criteria. For example, sort keys are designated where appropriate, ALIAS items are declared and singular sets chosen as VIA sets wherever these exist. Changes will probably be necessary before the final schema is decided upon and the outputs of the ADD system should assist the DBA in making these changes. These outputs comprise a copy of each run of the user interface, showing displays and responses; the SDM model specified; a list of all relations for the application; a DMS 1100 schema for the application; and a listing of what happened to each item in the SDM description. (This can be either: the sets, records or items created for it, with any integrity constraints, or "not implemented").

The relation schema synthesised by ADD is in 2NF and adheres to the principles of normal form theory by removing embedded dependencies wherever these are recognised. For this reason, the system was extended to include an optional component whereby FDs between attributes could be identified.

It is appropriate at this point to explain how the ADD system has been tested. Since it was designed in a modular fashion, each aspect of the design was fully tested before continuing with the next. This involved many test runs, as all the different types of

situation under which a modelling feature could be used, were tested. This process proved to be worthwhile, as it resulted in two earlier versions of ADD being adjusted to arrive at the present system. The SDM example given in Hammer and McLeod's article [Hammer 1981] was used in its entirety as the final test, since it includes all modelling constructs used at least once. The results of this example are given in appendices F,G and H. To test the user interface program, a computer novice was asked to experiment with ADD (in addition to my own numerous runs). First the program, and then its displays, were modified as a result of these experiments.

The advantages of ADD can be summarised as follows. It is helpful to systems analysts and DBAs in that much of their time usually spent in meeting with the user, or in coding the basic skeleton of the schema, is saved. Further, the database generated will form a good starting point for the process of database evaluation and tuning, as its logical structure has been carefully chosen. The system is original in that unlike the other automated database design systems [Gerritsen 1975,Hubbard 1979,Wang 1975] it is based on a semantic data model, not on processing requirements or functional dependencies.

ADD also deals with a semantically richer requirements specification, and unlike the others, automates the process of requirements gathering as well. It also differs from the systems of Hubbard and of Wang and Wedekind in that it designs a Codasyl DBTG structure.

Although the network schema generated is specifically in the DDL of DMS 1100, this is so close to the CODASYL specifications that changes to accomodate other CODASYL DBMS's would be minimal.

10.2. Possible changes and extensions.

There are some modifications which could be considered for ADD. For example, It may be argued that a DMS 1100 database should not have been used for storing the evolving network design, as this affects the system's portability. However, only the QDB-manipulation subroutine would need to be altered if a different database or data structure were used. As each call involves a very simple operation, altering this subroutine should be easily done. It would in fact be trivial if the system were used to design databases for another CODASYL DBMS. The QDB and its associated subroutine would require very few changes, because DMS 1100 adheres so strictly to the DBTG recommendations [Codasyl 1971, Codasyl 1977]. This small amount of effort that may be necessary is more than compensated for by having the versatility of a database for storing the structure being created. In fact, if ADD were developed on another machine, its database system would almost certainly have been used for this purpose, because of the great extent to which it simplifies the system.

It became apparent, when immersed in the difficulties created by attribute derivations and subclass definitions, that there is definitely a trade-off between efficiency and integrity. It might be argued that some integrity checks could have been omitted. This point is clearly debatable. One of the principles on which ADD is based is the following: if something forms part of the SDM specification, then it is sufficiently important to be included in the synthesised database. Furthermore, it was felt that decisions as to what was sufficiently unimportant to be able to ignore should not be made by an automated process, but by man.

Although ADD does not design a hierarchical database, this could fairly easily be incorporated into the system because a hierarchy is essentially a special type of network. Integrity constraints would need to be incorporated according to the hierarchical DBMS involved, as would the handling of many:many relationships.

The present system could be extended by improving the physical aspect of the design, which would mean including another module to obtain the relevant data volume and processing details from the user. The system was initially envisaged as a design tool for conceptual schema design only. Thus the algorithm choosing physical criteria for the database aimed at the simplest, most straight forward solutions. The system could also be extended to build up multiple external views of an enterprise and then consolidate these into one global logical model. This however is a separate area of its own, and hence was not considered. It can be said with some certainty that the inclusion of this facility would greatly complicate the model-creation phase of ADD. The user interface could also be altered so as to incorporate some natural language understanding, or to provide more levels of 'help'. The ADD system could form a basis for future research, with automated loaders, query-language generators and programs to generate different subschemas being a natural extension to the system.

10.3. Design principles emerging from this study.

A great deal has been learnt in undertaking to write this automatic database designer. The importance of a good user interface was realised at the outset, and the high amount of care

that needs to be expended on such a program became evident when testing the system with non-DP persons.

The ADD system development clearly showed the ease with which a relational database can be designed, in comparison with a CODASYL one. This is probably due to the simplicity of the relational model, its high degree of data independence (no access paths need be specified) and the guidelines that exist in normal form theory. However it should be noted that the question of integrity was not addressed because there is no relational DBMS on the Univac 1108. Since most of the complexity in converting an SDM specification to a computer-compatible schema arises when incoporating integrity assertions, their inclusion may however increase the difficulties of relational schema design.

The usefulness of data models and the importance of one's choice of data model was evident early in the design of the system, and increasingly so in the latter stages of its development. The main problem with SDM was seen to be its incompatibility with CODASYL databases. Hence one can deduce that a model should be restricted in its use so that it remains within the scope of Database Management Systems and should not include features unimplementable on conventional systems. Some general principles for data model design are presented in the next section.

10.4. Design criteria for Data Models.

Having studied the various data models currently available, and implemented the ADD system, the following criteria for good data models are suggested. A data model is best used by the personnel of the organisation requiring the database. Only they truly

understand this enterprise and its data, in a way that systems analysts cannot, except possibly after much effort. Even then, users must still be able to understand the model. For this reason, a data model should be simple and sufficiently easy to be used by computer-naive persons, possibly with some assistance from the analyst. Its syntax should be straight forward, avoiding complicated languages such as predicate calculus. Models should also not be too DP-like, as is DAPLEX for example [Shipman 1981].

For a model to be easy to conceptualise and work with, entities and attributes should be the central modelling elements, with relationships playing only an auxiliary role. If the user can specify his model by providing a list of all entities of interest, and then describe these by means of attributes, a correct and complete model should be obtained. Relationships can then be regarded either as entities or as attributes. To keep the complexity of the model within the capabilities of a non-computer user, the model should not be based on FDs (or MVDs). These can always be identified by (or under the guidance of) a systems analyst once the model is complete.

To increase the ease with which the model can be used to describe the real world, there should be many modelling tools and the designer should be allowed a free interpretation of his environment. However, the ease with which the model is conceptualised should not be compromised, nor should any complicated constructs be introduced. Thus, semantically expressive modelling elements for stating integrity assertions are desirable, as is the concept of generalisation, which will be applicable to relationships (as entities) as well. A name-based model is advocated, as this in conducive to a clearer view of a

model and forces the user to think carefully, lessening the probability of incorrect definitions. Similarly, entities should represent themselves, and the concept of keys should be de-emphasised, because they are not 'natural' to users [Martin 1973].

Dynamics should not be part of a data model, as this results in a database based on functional requirements, rather than on inherent characteristics of data; furthermore dynamics constructs cannot be incorporated into conventional schemas. This last point is one of the most important to bear in mind when designing a data model. If it is not compatible with available DBMS's, but provides modelling elements beyond their scope, then its suitabiltiy for database design is jeopardised.

10.5. Criteria for Database Design Methodologies.

Experience with the ADD system has also indicated the following criteria for database design methodologies. Above all, the user should partake in as much of the design process as is possible. For this reason, the use of a data model for initial description of the enterprise is advocated, as this enables a precise specification to be made in a natural way by computer-naive people. The major phases of the methodology should thus be:

1) construction of a data model of the enterprise by the user;
2) conversion of this data model to a logical database structure;
3) incorporation of physical, integrity and security parameters;
4) repeated evaluation and refinement.

It should be noted that the construction of a prototype (which results at the end of step 3), is suggested as essential in any

database design. It is only through experimentation with a prototype that misconceptions and overheads can be identified. The methodology should also ensure that the transitions from one phase to the next are smooth and well-defined.

The first phase, requirements gathering, is notably concerned only with data and its characteristics, and not with the transactions of the enterprise. Factors such as processing and data volumes should not be considered until phase three, when deciding the physical structure. During the initial stage, a data dictionary is helpful, and a method of handling multiple views and then consolidating these into a global model, should be devised.

A methodology is of very little assistance to a designer unless it states explicitly how to perform phase two. Existing methodologies still leave the creation of a logical structure largely in the hands of the systems analyst, providing little more than guidelines for this important and difficult step. If a data model has been used to describe the enterprise, then the logical design phase can be made more concrete. Each of the modelling constructs can be considered and rules given for how to convert these into structures compatible with database systems. The concepts of normal form theory should be applied to the records or relations obtained at the end of phase two. It is suggested that 3NF or Elementary Key Normal Form [Zaniolo 1982] is adequate, otherwise too many simple records or relations may result, thus affecting efficiency.

The assignment of physical parameters to a given logical structure has been somewhat neglected by database researchers. A good methodology should at the very least provide guidelines for this, and highlight problem areas where inefficiency can arise. As a

final objective, a database design methodology should be adaptible to automation, if only in part. This is considered desirable because the (partial) automation of the design process is certain to be a reality in the future [Buchman 1979].

The entire process cannot be automated; the analytical powers, intuition and creativity of the human mind are an essential part of designing. All that can be done is to automate as much of the design process as is reasonable, to alleviate the task of the human(s) involved. The ADD system is a step in this direction.

[Buchman 1979] Buchman, A.P. and Dale, A.G. "Evaluation Criteria for Logical Database Design Methodologies", Computer-Aided Design, Vol. 11, No. 3, may 1979, pp. 121-126.

[Codasyl 1971] Codasyl Data Base Task Group Report, Conf. on Data Systems Languages, ACM New York NY, 1971.

[Gerritsen 1975] Gerritsen, R. "A Preliminary System for the Design of DBTG Data Structures", Comm. of the ACM, Vol. 18, 1975, pp. 551-557.

[Hammer 1981] Hammer, M. and McLeod, D. "Database Description with SDM: A Semantic Database Model", ACM Trans. on Database Systems, Vol. 6, No. 3, 1981, pp. 351-386.

[Roussopoulos 1979] Roussopoulos, N. "Tools for Designing Conceptual Schemata of Databases", Computer-Aided Design Vol. 11, No. 3, May 1979, pp. 119-120.

[Shipman 1981] Shipman, D. "The Functional Data Model and the Data Language DAPLEX", ACM Trans. on Database Systems, Vol. 6 No. 1, March 1981, pp. 140-173.

[Wang 1975] Wang, C. and Wedekind, H. "Segment Synthesis in Logical Database Design", IBM Jounal of Research and Development, Vol. 19, January 1975, pp. 71-77.

[Zaniolo 1982] Zaniolo, C. "A New Normal Form for the Design of Relational Database Schemata", ACM Trans. on Database Systems, Vol. 7, No. 3, 1982, pp. 489 - 499.

11. APPENDIX A. DATABASE SYSTEMS.

Definitions of the term "database system" vary from "nothing more than a computer-based record-keeping system" (Date 1981] to "a generalised collection of data which is structured on natural data relationships so that it provides all necessary access paths to each unit of data in order to fulfil the differing needs of all users" [Deen 1977]. The power of a database system lies with its DBMS (Database Management System), "a set of programs that operates on the database in accordance with the user's commands" [Kroenke 1977]. The DBMS is a means of centralising physical and conceptual control of the data and its usage. A DBMS enhances the quality of the data - maintaining its integrity, consistency privacy and security. It can also synchronise requests in a multi-user environment and can provide a high degree of data independence. Data independence is the insensitivity of application programs using a common database to restructuring of the data - i.e. programs do not depend on any one particular storage structure or access strategy. There are numerous other advantages to using a database system, such as simplicity, flexibility, protection and recovery procedures and elimination of uncontrolled redundancy. In addition, most database systems provide a query language facility for direct interrogation of the database without the need for an application program. Database systems aim at reducing processing time and storage space: this however depends on the database design, programmer expertise and the nature of the organisation using the database; results will vary accordingly.

The three 'views' of a database, and then the three types of database system, are briefly described. Thereafter a short

comparison of these three approaches is made.

B1. Views.

IN ANSI/SPARC [Tsichritzis 1978] terminology, an external view is one individual's view of the database. Usually there are several differing external views of a database, since users generally work with only a section of the database, and may not visualise shared data in the same way. The conceptual view is a view of the entire information content of the database, in other words it is a consolidation of all the external views. In contrast, the internal view is the view of the entire database as it is actually physically stored. Ideally, only the DBMS should be concerned with the latter, and not the application programmers. The conceptual view is specified by writing a schema, as was explained in chapter two of this thesis. This task is performed by or in conjunction with the Database Administrator, who is also responsible for database maintenance. The data definition language almost invariably has statements that describe the physical layout of the database. Although it conflicts with the aim of data independence, the efficiency of the system can be improved by taking advantage of this. Security and integrity constraints are also given in the schema. Subschemas, as outlined in chapter two, represent different views of the database, and are derivable from, and usually a subset of, the schema.

B2. Network Database Systems.

In 1971 the Database Task Group (DBTG) of the Conference on Data Systems Languages (CODASYL) [Codasyl 1971] laid down proposals for a network database system, which included specifications of the network model, and of notations for a DDL, SDDL and DML. The network model represent entities as records, and relationships as

sets.

Sets are the major CODASYL innovation. In the schema, they require an explicit definition and have properties of their own. A set type is described as having a certain record type as owner and another record type as member. A set represents a 1:many relationship or mapping from the owner to the member (or 1:1, as a special case). Several member record types can participate in one set type, and two or more set types with owner R1 and member R2 can be defined. Many-to-many relationships, and associations involving only one record type, cannot be directly implemented in a network database. Although there are ways of circumventing these problems, they tend to make the network intricate and thus more difficult to work with. A description of a database in the schema DDL consists of four major sections : an introductory clause, one or more area clauses, one or more record clauses, one or more set clauses. The introductory clause is used ot name the database and to state certain global security and integrity constraints. An area is a subdivision of the database, which in many implementations corresponds to a file. These sections have already been explained in the body of the thesis.

An interesting aspect of the CODASYL schema is that it includes the specification of access strategy and storage structure, which are physical considerations.

B3 Hierarchical database systems.

In a hierarchical database system, an entity is represented by a segment, which is analogous to a 'record' in the network model. That is, it is the unit of data access and comprises a number of fields. Relationships between segments are represented by means of rooted trees or hierarchies. A tree is a structure in which every segment has one and only one parent(analagous to 'owner' in

the DBTG model), except for one segment type, the root, which has no parent. A parent:child association represents a 1:many mapping from parent to child. Unlike network models, a segment occurrence can only be linked to one occurrence of its parent. For example, in a CODASYL schema, two sets ROUTE->SHIP can be defined, to represent the current and previous routes of each ship. In the hierarchical model this could not be implemented directly in this way, because a SHIP occurrence could only have one ROUTE occurrence as parent. Every segment has one or more fields designated as its key, and children are linked to their parent in key sequence order. This factor is utilised in the DML, which includes commands such as 'get next', meaning next with respect to this hierarchical key sequencing. Many-to-many relationships between segments cannot be directly represented in a hierarchical database; however several commercial hierarchical DBMSs provide a means of circumventing this problem. IBM's IMS for example uses the concept of 'virtual pairing'. This involves using two tree types, where the child segment of each tree acts as a pointer to the appropriate data in the other tree type. The hierarchical model uses different terminology from that of the CODASYL DBTG: The schema comprises one DBD (Database Definition) for each PDB (Physical Database); a subschema is termed a PSB (Program Specification Block) and constitutes one PCB (Program Communication Block) for each LDB (Logical Database) in that view.

A PCB can refer to a subsection of the DBD, in which case the segments it requires are called 'sensitive segments'. If a segment is non-sensitive, none of its "descendants" can be defined in that PCB.

B4. Relational databases.

The relational model was invented by E.F.Codd and differs from the

other models in that it does not have a structure of inter-linked records. Whereas relations can be viewed as record (segment) types, there are no inter-relation connections in a relational model. A relation may be formally defined as follows: Let D1,D2,...,Dn be n sets, not necessarily distinct. A relation is a collection of n-tuples of the form (d1,d2,...,dn) where each di is drawn from the corresponding set Di. D1 to Dn are called the domains of the relation, and n is its "degree". A relation is usually represented diagramatically as a table, with one column for each Di, and one row per tuple. Each column is called an 'attribute' and is given a name. A relational database or relation scheme consists of several relations, some representing objects or events and others representing relationships between objects. Thus the relational model is clearly simpler than the other two database models. If two or more objects are associated in some way, this is represented by having the keys of these objects appearing together in the relation defining that association. For example, to denote that SHIPS and OFFICERS are related, a relation R(SHIP#,OFFICER#) would be declared. Hence many-to-many relationships are as easy to specify as one-to-many. Another aspect of the relational model is that it is based on the idea of certain attribute(s) in every relation being designated as the key for that relation. This database system has been criticised for this by authors such as Kent [Kent 1977], who claim that keys such as OFFICER# or ENGINE# are artificial concepts and hence should be avoided. Due to the lack of inter-relation links, database queries involving more than one relation are formulated in terms of their common domains. This is the central principle of languages designed for relational systems. Such languages have a mathematical foundation, and are either based on the relational

algebra or relational calculus. An important feature of these languages is that they are non-procedural; data is accessed by specifying conditions on their values, and not by 'navigational' means.

B5. Conclusion.

The three models reviewed in this appendix are essentially all record-based. They differ in terms of how they represent relationships between these 'records'. The network model uses sets for this purpose; these define one-to-many relationships. 'Intermediate', dummy records are needed to specify many-to-many associations. However such a database system is fairly flexible as several relationships between identical record types can be defined. Unfortunately, this often results in a structurally intricate graph which is difficult to navigate. A greater degree of data independence exists in the relational model, where all associations are represented by means of common values in the relevant 'records' (tuples). However, this model relies on the specification of unique identifiers, as does the third type of model, the hierarchical. The latter is certainly the most restrictive of the three, as segments can only be linked to form trees, not graphs. As a result, its DML however is generally simpler than that of network systems. In conclusion, there are clearly benfits and disadvantages in all three types of database system. It would seem that the relational model is simplest and has the highest degree of data independence. For this reason, much research has been devoted to the development of relational database management systems.

[Codasyl 1971] Codasyl Committee on Data System Languages, Codasyl Database Task Group Report, ACM New York NY, 1971.

[Date 1981] Date, C.J. "An Introduction to Database Systems", (3rd

ed.), Addison-Wesley, 1981.

[Deen 1977] Deen, S.M. "Fundamentals of Database Systems", MacMillan Press, London, 1977.

[Kent 1977] Kent, W. "New Criteria for the Conceptual Model", in Systems for Large Databases, (eds. Lockemann and Neuhold), North-Holland, 1977, Proc. of the 2nd Int. Conf. on Very Large Databases, 1976.

[Kroenke 1977] Kroenke, D. "Database Processing: Fundamentals, Modeling, Applications", Science Res. Associates, Palo Alto CA, 1977.

[Tsichritzis 1978] Tsichritzis, D.C. and Klug, A. "The ANSI/X3/SPARC DBMS Framework. Report of the Study Group on Database management Systems", Inf. Syst. Vol. 3, 1978, pp. 173-191.

12. APPENDIX B. DMS 1100 DDL SYNTAX

The following DMS 1100 Data Definition Language Syntax Skeleton is taken from "Database Management System (DMS 1100)", UP 7909, Rev. 3A, Sperry Univac, St. Paul Minn., 1972.

DDL OUTLINE
- IDENTIFICATION DIVISION
- DATA DIVISION
  - DATA NAME SECTION
  - AREA SECTION
  - RECORD SECTION
  - SET SECTION

DDL SYNTAX SKELETON

IDENTIFICATION DIVISION.

SCHEMA NAME IS *schema-name* IN { TIP FILE *TIP-file-code* / FILE *file-name-1* }.

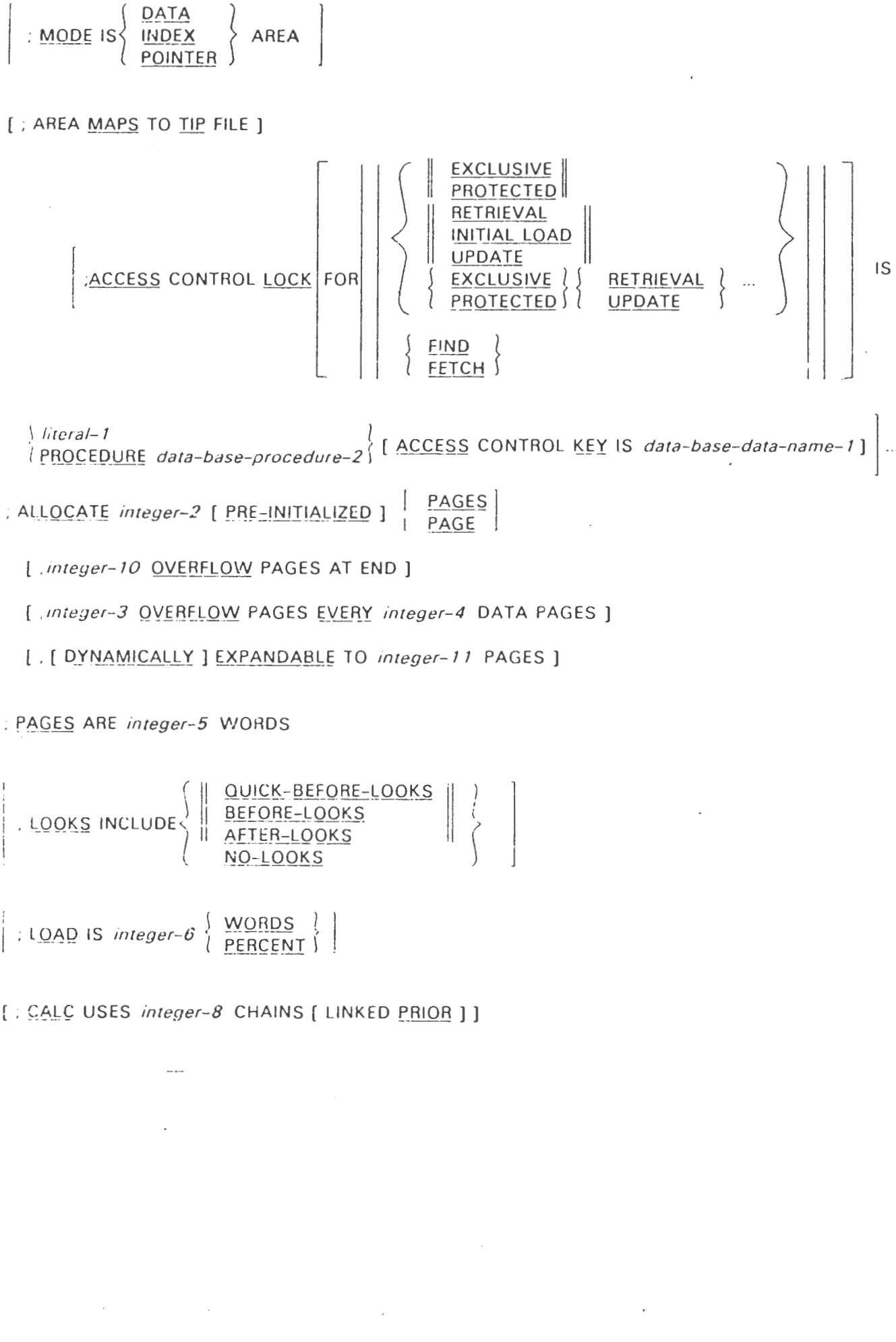
[SCHEMA QUALIFIER IS *qualifier-name*]

[LOCK FOR COPY IS *literal-1*]

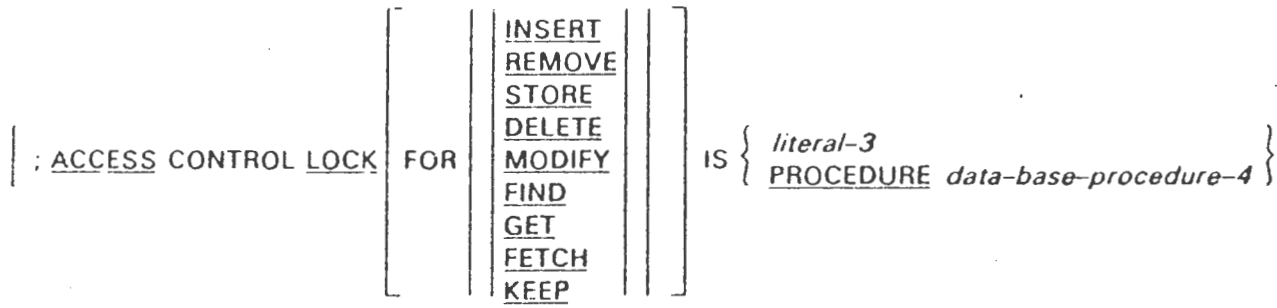
[ACCESS CONTROL LOCK [FOR [DMU / FREE / IMPART / LOG / DRU]] IS { *literal-2* / PROCEDURE *data-base-procedure-1* }]

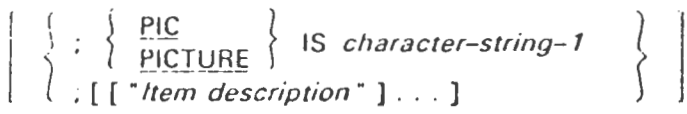
[[ACCESS CONTROL KEY IS *data-base-data-name-1*]] ...

```
DATA DIVISION.

[ DATA NAME SECTION.

  { 01 | 77 }  data-base-data-name-1

  [ ; { PIC | PICTURE } IS character-string-1 ]

  [ ; USAGE IS { COMPUTATIONAL
                 COMP
                 COMPUTATIONAL-1
                 COMP-1
                 COMPUTATIONAL-2
                 COMP-2
                 COMPUTATIONAL-4
                 COMP-4
                 DISPLAY
                 DISP
                 DISPLAY-1
                 DISP-1
                 DATABASE-KEY
                 AREA-NAME
                 AREA-KEY
                 RECORD-NAME
                 SET-NAME
                 ACCESS-KEY
                 ALIAS } ]

  [ ; DATA NAME CODE IS integer-1 ]

AREA SECTION.

[ AREA CONTROL IS integer-2 AREAS ]

[ AREA LOOKS INCLUDE { QUICK-BEFORE-LOOKS
                       BEFORE-LOOKS
                       AFTER-LOOKS
                       NO-LOOKS } ]

[ RECOVERY-POINTS ARE EVERY integer-3 BLOCKS . ]

AREA NAME IS area-name-1

[ ; AREA CODE IS integer-1 ]
```

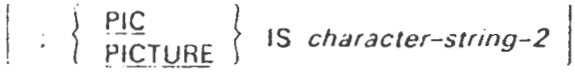

; MODE IS DATA INDEX POINTER AREA
[; AREA MAPS TO TIP FILE]
; ACCESS CONTROL LOCK FOR EXCLUSIVE PROTECTED RETRIEVAL INITIAL LOAD UPDATE EXCLUSIVE PROTECTED RETRIEVAL UPDATE ... FIND FETCH IS
literal-1 PROCEDURE data-base-procedure-2 [ACCESS CONTROL KEY IS data-base-data-name-1] ...
; ALLOCATE integer-2 [PRE-INITIALIZED] PAGES PAGE
[, integer-10 OVERFLOW PAGES AT END]
[, integer-3 OVERFLOW PAGES EVERY integer-4 DATA PAGES]
[, [DYNAMICALLY] EXPANDABLE TO integer-11 PAGES]
; PAGES ARE integer-5 WORDS
, LOOKS INCLUDE QUICK-BEFORE-LOOKS BEFORE-LOOKS AFTER-LOOKS NO-LOOKS
; LOAD IS integer-6 WORDS PERCENT
[; CALC USES integer-8 CHAINS [LINKED PRIOR]]

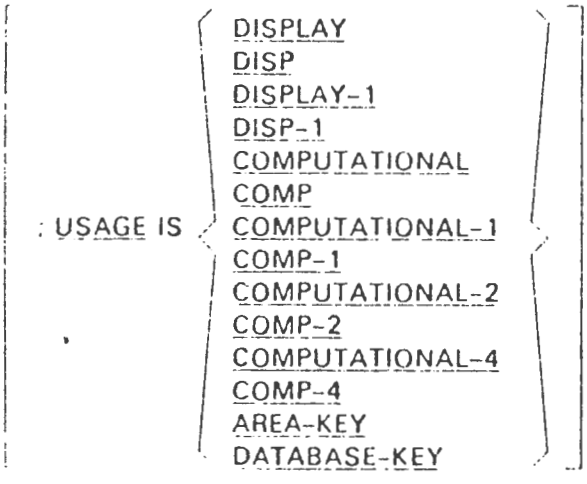
```
RECORD SECTION.

RECORD NAME IS record-name-1.

[ ;RECORD CODE IS integer-1 ]

                                  ⎧ DIRECT data-base-data-name-1
                                  ⎪     ,data-base-data-name-2
                                  ⎪ CALC data-base-procedure-name-1
                                  ⎪     [ IN data-base-data-name-3 ]
                                  ⎪     USING data-base-identifier-1
                                  ⎪         [ ,data-base-identifier-2 ]...
                                  ⎪     DUPLICATES ARE [ NOT ] ALLOWED
                                  ⎪ INDEX SEQUENTIAL
: LOCATION MODE IS                ⎪     USING { ASCENDING  } [ RANGE ] KEY
                                  ⎪           { DESCENDING }
                                  ⎪         data-base-identifier-3
                                  ⎪         [,data-base-identifier-4]...
                                  ⎪     INDEX AREA IS area-name-1
                                  ⎪     [ LINKS ARE NEXT [ AND PRIOR ] ]
                                  ⎪     DUPLICATES ARE [ NOT ] ALLOWED
                                  ⎪     [ INTERVAL IS integer-2 PAGES ]
                                  ⎪ VIA set-name-1 SET
                                  ⎩   [ INTERVAL IS integer-3 PAGES ]

[ { : WITHIN area-name-2 [ { integer-4        } ] } ... ]
  {                        { data-base-data-name-4 }   }
  {   { THRU    }  { integer-5        }   |     }
  {   { THROUGH }  { data-base-data-name-5 }   |     }

[ . RESERVE integer-6 POINTERS ]

[ : RECORD MODE IS { FIELDATA } COBOL ]
                   { ASCII    }

[ [ : CHECK IS || PICTURE                          || ] ]
               || DATA-TYPE                        ||
               || PROCEDURE data-base-procedure-2  ||

[ . FOR ENCODING CALL data-base-procedure-3
  FOR DECODING CALL data-base-procedure-4 ]
```





■ Item Description Syntax Skeleton:





[*"result subentry"*]

RESULT SUBENTRY SYNTAX

Format-1:

```
[ ; RESULT OF SUM ON THIS RECORD USING data-base-identifier-11
      , data-base-identifier-12 ] . . . ]
```

Format-2:

```
[ ; RESULT OF PROCEDURE data-base-procedure-5 ON THIS RECORD
     [ USING data-base-identifier-13 [ , data-base-identifier-14 ] . . . ] ]
```

Format-3:

```
RESULT OF

{ COUNT ON | ALL MEMBERS OF SET | set-name-2 | ...                                  | }
{          | SET set-name-3 MEMBER | record-name-2 | ... | ...                    | }
{ SUM ON | SET set-name-4 MEMBER record-name-3 USING data-base-identifier-15       | }
{          [ , data-base-identifier-16 ] ... | ...                                 | }
```

Format-4:

```
RESULT OF PROCEDURE data-base-procedure-6

ON | ALL MEMBERS OF SET | set-name-5 [ USING MEMBERSHIP ONLY ] | ...                  |
   | SET set-name-6 | MEMBER record-name-4                                          |
   |   USING { data-base-identifier-17 [ , data-base-identifier-18 ] ... } } ... } | ... |
   |         { MEMBERSHIP ONLY                                                     |     |
```

```
[ ; CHECK IS | DATA-TYPE                                                                                    |
             | VALUE [NOT] { NULL                                                                         |
             |             { literal-3 [THRU literal-4] [ , literal-5 [THRU literal-6] ] ... }             |
             |               MIN literal-2                                                                |
             |               MAX literal-2                                                                |
             | PROCEDURE data-base-procedure-7                                                            |
```

```
[ ; FOR ENCODING TO n { CHARS } { CALL data-base-procedure-8                                  }
                      { BITS  } {   FOR DECODING CALL data-base-procedure-9                  }
                      {       } { USE table-name-1                                           }
                      {       } { REPLACE literal-7 WITH literal-8                           }
                      {       } {   [ , REPLACE literal-9 WITH literal-10 ] ...              } ]
```

```
|                                   [     || FETCH  || ]        { literal-8                          }
| ; ACCESS CONTROL LOCK [ FOR || STORE  || ] IS { PROCEDURE data-base-procedure-14 }
|                                   [     || GET    || ]
|                                   [     || MODIFY || ]

[ ACCESS CONTROL KEY IS data-base-data-name-10 ] | . . .

SET SECTION.

SET NAME IS set-name-1

[ ; SET CODE IS integer-1 ]

; MODE IS { CHAIN [ LINKED PRIOR ]                              }
          { POINTER ARRAY POINTER AREA IS area-name-1 }

              { FIRST                                                                 }
              { LAST                                                                  }
              { NEXT                                                                  }
              { PRIOR                                                                 }
              { SORTED [ INDEXED INDEX AREA IS area-name-2 ]                          }
; ORDER IS    { [ BY DATABASE-KEY                                              ]      }
              { [ WITHIN RECORD-NAME                                           ]      }
              { [ BY DEFINED KEYS                                              ]      }
              { [       [                   { FIRST          } ]               ]      }
              { [       [ DUPLICATES ARE    { LAST           } ]               ]      }
              { [       [                   { NOT ALLOWED    } ]               ]      }
              { [       [                   { SYSTEM ORDERED } ]               ]      }

|                                   [     || INSERT  || ]
|                                   [     || REMOVE  || ]        { literal-2                         }
| ; ACCESS CONTROL LOCK [ FOR || ACQUIRE || ] IS { PROCEDURE data-base-procedure-1 }
|                                   [     || FIND    || ]
|                                   [     || FETCH   || ]

[ ACCESS CONTROL KEY IS data-base-data-name-9 ] | . . .

; OWNER IS record-name-1

    | "Member subentry" | . . .
```

Syntax Format of Member Subentry:

```
; MEMBER IS record-name-2 { AUTOMATIC | MANUAL } [ LINKED TO OWNER ].

[ { { ASCENDING | DESCENDING } [ RANGE ] KEY IS || RECORD-NAME
                                                 || data-base-identifier-1
                                                    [ ,data-base-identifier-2 ] ... || } ...

    [ ; DUPLICATES ARE { FIRST | LAST | NOT ALLOWED | SYSTEM ORDERED } ] ]

[ ; ACCESS CONTROL LOCK [ FOR || INSERT | REMOVE | FIND | FETCH || ] IS

  { literal-2 | PROCEDURE data-base-procedure-2 } [ACCESS CONTROL KEY IS data-base-data-name-2 ] ]...
```

| *"Set occurrence selection subentry"* |

Syntax Format of Set Occurrence Selection Subentry

Format-1:

```
SET OCCURRENCE SELECTION IS THRU

{ CURRENT OF SET | LOCATION MODE OF OWNER }

[ { USING data-base-identifier-3 [ ,data-base-identifier-4 ] ...
    [ ALIAS { FOR data-base-identifier-5 | OF data-base-data-name-1 } IS data-base-data-name-2 ] ... } ]
```

Format-2:

```
; SET OCCURRENCE SELECTION IS THRU set-name-2 USING

{ CURRENT OF SET | LOCATION MODE OF OWNER }

[ { USING data-base-identifier-6 [ ,data-base-identifier-7 ] ...
    [ ALIAS { FOR data-base-identifier-8 | OF data-base-data-name-3 } IS data-base-data-name-4 ] ... } ]

{ set-name-3
  { USING data-base-identifier-9 [ ,data-base-identifier-10 ] ...
  | ALIAS FOR data-base-identifier-11 IS data-base-data-name-5 | ... } } ...
```

Format-3:

```
: SET SELECTION [ IS THRU set-name-4 ]

OWNER IDENTIFIED BY

{ CURRENT OF SET
  LOCATION MODE
    [ AREA-KEY data-base-data-name-6 [ EQUAL TO data-base-data-name-7 ]
      AREA-NAME data-base-data-name-8 [ EQUAL TO data-base-data-name-9 ]
      KEY | data-base-identifier-12 [ EQUAL TO data-base-data-name-10 ] | ... ]
  MEMBERSHIP IN set-name-5
    [ KEY | data-base-identifier-13 [ EQUAL TO data-base-data-name-11 ] | ... ] }

[ THEN THRU set-name-6 OWNER IDENTIFIED BY
    KEY | data-base-identifier-14 [ EQUAL TO data-base-data-name-12 ] | ... ] ...

[ TABLE SECTION
  { TABLE NAME IS table-name-1 [ FOR ENCODING TO n { BITS | CHARS } ]
  | REPLACE literal-1 WITH literal-2 [ , REPLACE literal-3 WITH literal-4 ] ... } ... ]
```

13. APPENDIX C. DMS 1100 FORTRAN DML SYNTAX

The following DMS 1100 FORTRAN Data Manipulation Language Syntax Skeleton is taken from "Database Management System (DMS 1100)", UP 7909, Rev. 3A, Sperry Univac, St. Paul Minn., 1972.

■ FDMLP Syntax

FDML declarations precede all executable statements.

RECORD NAME *record-name-1 , . . .*

SET NAME *set-name-1 , . . .*

REALM NAME *realm-name-1 , . . .*

KEY *key-name-1 , . . .*

REALM KEY *realm-key-name-1 , . . .*

ACQUIRE LIST *key-count-name key-list-name(integer)*

EXTERNAL *subroutine-name-1 , . . .*

INVOKE ([SUBSCHEMA =] *file-name.subschema-name* , SCHEMA= *schema-name*

[INVKEY = *invoke-key*
ENVIRONMENT = HVTIP
RUNID = *run-unit-name*
PRIORITY = *integer-1*
POINTERS = *integer-2*
INCLUDE = STATISTICS
COMMON [=] [/ [*common-name*] /]
EQUIVALENCE [=] { ALL | (*record-name-1, record-name-2* [, *record-name-3*] . . .) }
RECOVERY = { COMMAND | RUN }
ERRITEMS = *integer-3*
{ ERR | ERROR } = *action*]

)

where *action* has the following format:

```
{ statement-number-1
{ subroutine-name [ ( { parameter-name-1 } ... ) ] }
                  { literal-1            }
                  { { $ } statement-number-2 }
                  { { & }                    }
```

■ FDML Commands

```
                  { CURRENCY = key-name , { RECORD = record-name }
                  {                       { REALM = realm-name  }
                  {                       { SET = set-name      }
                  {                       { RUN UNIT            }
ACCEPT (          {
                  { { REALM KEY = realm-key-name } , { RECORD = record-name }
                  { { REALM NAME = realm-name    }   { REALM = realm-name   }
                  {                                  { SET = set-name       }
                  {                                  { KEY = key-name-2     }
                  {                                  { RUN UNIT             }

[ , { ERR   } = action ] )
    { ERROR }
```

Key-expression Use And Functions:

key-name = *key-expression*

realm-key-name = REALM KEY ({ KEY = *key-name* })
                               { *key-expression* }

realm-name = REALM NAME ({ KEY = *key-name* })
                           { *key-expression* }

where *key-expression* is:

CURRENCY ({ RECORD = *record-name* }
           { REALM = *realm-name* }
           { SET = *set-name*   }
           { RUN UNIT   })

ACQUIRE (*integer-name* , *key-list-name* [, START = *key-name*] , SET = *set-name*

[, DEFINED] [, { END = *action-1* }])
                  { { ERR   } = *action-2* }
                  { { ERROR }   }

```
CONNECT ( [ RECORD = record-name , ] { ALL
                                     { SET = set-name
                                     { SET = ( set-name-1 , . . . ) }
    [ , { ERR   } = action ] )
        { ERROR }
```

```
DEPART [ ( || ROLLBACK                  || ) ]
       [ ( || { ERR   } = action        || ) ]
              { ERROR }
```

```
DISCONNECT ( [ RECORD = record-name , ] { ALL                          }
                                        { SET = { set-name             } }
                                                { ( set-name-1 , . . . ) }
    [ , { ERR   } = action ] )
        { ERROR }
```

```
ERASE [ ( || RECORD = record-name || ) ]
      [ ( || { ALL       }       || ) ]
             { PERMANENT }
      [ ( || { ERR   } = action  || ) ]
             { ERROR }
```

```
{ FIND  } ( rse [ , || retaining-clause                  || ] )
{ FETCH }       [ , || END = action-1                    || ]
                    { ERR   } = action-2
                    { ERROR }
```

where *rse* has the following format:

"rse-1":

```
KEY = { key-name .
      { realm-key-name [ , REALM = realm-name ] } [ , RECORD = record-name ]
```

"rse-2":

```
[ ANY ,       ] RECORD = record-name
[ DUPLICATE , ]
```

"rse-3":

```
DUPLICATE [ , RECORD = record-name ] , SET = set-name , USING =

    { item-name
    { ( item-name-1 , . . . ) }
```

"rse-4":

{ **NEXT** | **PRIOR** | **FIRST** | **LAST** | **ORDINAL** = *integer-name* } , ‖ **RECORD** = *record-name* | { **SET** = *set-name* | **REALM** = *realm-name* } ‖

"rse-5":

CURRENT , ‖ **RECORD** = *record-name* | { **SET** = *set-name* | **REALM** = *realm-name* } ‖

"rse-6":

OWNER , **SET** = *set-name*

"rse-7":

RECORD = *record-name* , **SET** = *set-name* [, **USING** = { *item-name* | (*item-name-1* , . . .) }]

FINISH ({ **ALL** | **REALM** = *realm-name* | **REALM** = (*realm-name-1* , . . .) } [, { **ERR** | **ERROR** } = *action*])

FREE [({ **ERR** | **ERROR** } = *action*)]

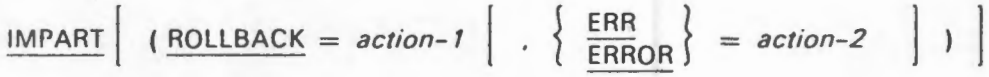
GET [(‖ **RECORD** = *record-name* | { **ERR** | **ERROR** } = *action* ‖)]

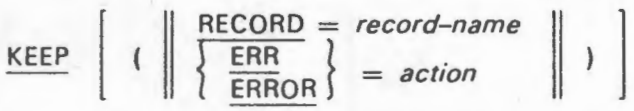
IF *format-1:*

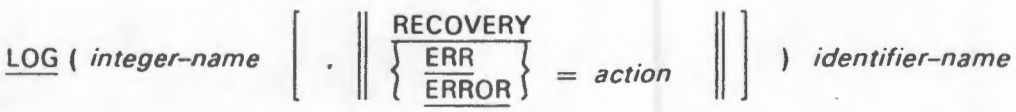
IF ([**.NOT.**] **EMPTY** (**SET** = *set-name*)) { *FDML-command* | *FORTRAN-statement* }

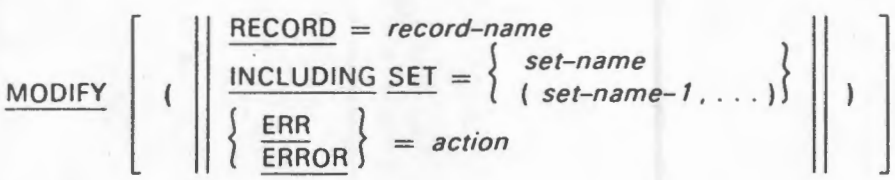
IF *format-2:*

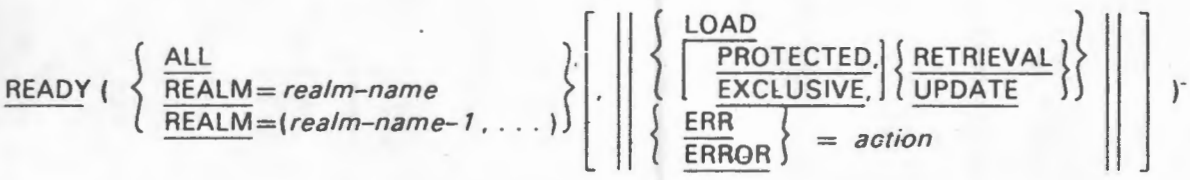
IF ([**.NOT.**] { **OWNER** | **MEMBER** | **TENANT** } ({ **SETS** = **ANY** | **SET** = *set-name* })) { *FDML-command* | *FORTRAN-statement* }

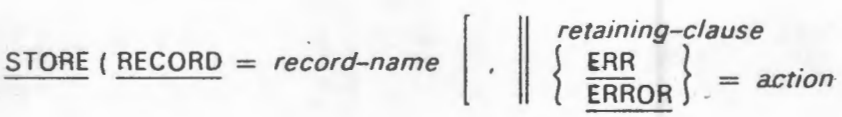

IMPART (ROLLBACK = action-1 . ERR ERROR = action-2)

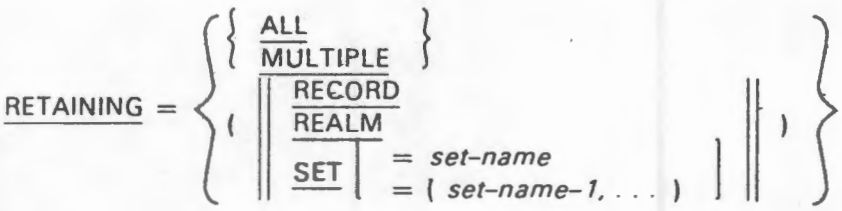

KEEP (RECORD = record-name ERR ERROR = action)


LOG (integer-name . RECOVERY ERR ERROR = action) identifier-name


MODIFY (RECORD = record-name INCLUDING SET = set-name (set-name-1, . . .) ERR ERROR = action)


READY (ALL REALM = realm-name REALM = (realm-name-1, . . .) . LOAD PROTECTED, EXCLUSIVE, RETRIEVAL UPDATE ERR ERROR = action)


STORE (RECORD = record-name . retaining-clause ERR ERROR = action)


RETAINING = ALL MULTIPLE (RECORD REALM SET = set-name = (set-name-1, . . .))

14. APPENDIX D. AN SDM MODEL OF A NAVAL ENVIRONMENT.

```
SHIPS
   description:all ships with potentially hazardous cargoes
  member attributes:
     Name
       valueclass: SHIP-NAMES
     Hull-Number
       valueclass: HULL-NUMBERS
        may not be null
       not changeable
     Type
       description: the kind of ship, for example merchant or
              fishing
       valueclass: SHIP-TYPE-NAMES
     Country-of-Registry
        valueclass: COUNTRIES
       inverse: Ships-Registered-Here
     Name-of-Home-Port
       valueclass: PORT-NAMES
     Cargo-Types
       description: the type(s) of cargo the ship can carry
       valueclass: CARGO-TYPE-NAMES
       multi-valued
     Captain
       description: the current captain of the ship
       valueclass: OFFICERS
       match: Officer of ASSIGNMENTS on Ship
     Shipengines
       valueclas: ENGINES
       multi-valued with size between 0 and 10
        exhausts valueclass
       no overlap in values
     Incidents-Involved-In
       valueclass: INCIDENTS
       inverse: Involved-Ship
       multi-valued
  identifiers:
     Name
     Hull-Number

INSPECTIONS
  description: inspections of oil tankers
  member attributes:
     Tanker-Inspected
        description: the tanker inspected
        valueclass: OILTANKERS
        inverse: Inspections-of-this-Tanker
     Inspection-Date
        valueclass: DATES
     Order-for-Tanker
        description: the ordering of the inspectons for a tanker
                     with the most recent inspection having value
                     1
        valueclass: INTEGERS
        derivation: order by decreasing Inspection-Date
```

```
                                within Tanker-Inspected
  class attributes:
     Number-of-Inspections
        description: the number of inspections in the database
        valueclass: INTEGERS
        derivation: number of members in this class
  identifiers:
     Tanker-Inspected + Inspection-Date

COUNTRIES
  description: countries of registry for ships
  member attributes:
     Country-Name
       valueclass: COUNTRY-NAMES
     Ships-Registered-Here
        valueclass: SHIPS
        inverse: Country-of-Registry
        multi-valued
  identifiers:
     Country-Name

OFFICERS
  description: all certified officers of ships
  member attributes:
     Officer-Name
        valueclass: PERSON-NAMES
     Country-of-Licence
        valueclass: COUNTRIES
     Date-Commissioned
        valueclass: DATES
     Seniority
        valueclass: INTEGERS
        derivation: order by Date-Commisssioned
     Commander
        description: the officer in direct command of this officer
        valueclass: OFFICERS
     Superiors
        valueclass: OFFICERS
        derivation: all levels of values of Commander
        inverse: Subordinates
        multi-valued
     Subordinates
        valueclass: OFFICERS
        inverse: Superiors
        multi-valued
     Contacts
        valueclass: OFFICERS
        derivation: where is in Superiors or is in Subordinates
  identifiers:
     Officer-Name

ENGINES
  description: ship engines
  member attributes:
     Serial-Number
        valueclass: SERIAL-NUMBERS
     Kind-of-Engine
        valueclass: ENGINE-TYPE-NAMES
  identifiers:
```

```
    Serial-Number

INCIDENTS
  description: accidents involving ships
  member attributes:
     Involved-Ship
        valueclass: SHIPS
        inverse: Incidents-Involved-In
   Incident-Date
        valueclass: DATES
    Description
        description: textual explanation of the accident
        valueclass: INCIDENT-DESCRIPTIONS
     Involved-Captain
        valueclass: OFFICERS
  identifiers:
     Involved-Ship + Incident-Date + Description

ASSIGNMENTS
  description: the assignments of captains to ships
  member attributes:
     Officer
        valueclass: OFFICERS
     Ship
        valueclass: SHIPS
  identifiers:
     Officer + Ship

OILTANKERS
  description: oil-carrying ships
  interclass connection: subclass of SHIPS where Cargo-Types
                         contains 'OIL'
  member attributes:
     Hull-Type
        description: specification of single or double hull
        valueclass: HULL-TYPE-NAMES
     Is-Tanker-Banned
        valueclass: YES/NO
        derivation: if in BANNED-SHIPS
     Inspections-of-this-Tanker
        valueclass: INSPECTIONS
        inverse: Tanker-Inspected
        multi-valued
     Number-of-times-Inspected
        valueclass: INTEGERS
        derivation: number of unique members in
                    Inspections-of-this-Tanker
     Last-Inspection
        valueclass: MOST-RECENT-INSPECTIONS
     Last-Two-Inspections
        valueclass: INSPECTIONS
        derivation: subvalue of Inspections-of-this-Tanker
                    where Order-For-Tanker <= 2
        multi-valued
     Date-Last-Examined
        valueclass: DATES
        derivation: same as Last-Inspection.Inspection-Date
     Oilspills-Involved-In
        valueclass: INCIDENTS
```

```
        derivation: subvalue of Incidents-Involved-In
                    where is in OILSPILLS
        multi-valued
  class attributes:
     Absolute-Top-Legal-Speed
        valueclass: KNOTS
     Top-Legal-Speed-in-Miles-per-Hour
        valueclass: MILES-PER-HOUR
        derivation: = Absolute-Top-Legal-Speed / 1.1

RURITANIAN-SHIPS
  interclass connection: subclass of SHIPS where
                         Country-of-Registry.Country-Name =
                         'RURITANIA'

RURITANIAN-OILTANKERS
   interclass connection: subclass of OILTANKERS where
                         Country-of-Registry.Country-Name =
                         'RURITANIA'

MERCHANT-SHIPS
  interclass connection: subclass of SHIPS where
                         Type = 'MERCHANT'

OILSPILLS
  interclass connection: subclass of INCIDENTS where
                         Description = 'OIL SPILL'
  member attributes:
     Amount-Spilled
        valueclass: GALLONS
     Severity
        valueclass: GALLONS
        derivation: Amount-Spilled / 100 000
  class attributes:
     Total-Spilled
        valueclass: GALLONS
        derivation: sum of Amount-Spilled over members of this
                    class

MOST-RECENT-INSPECTIONS
  interclass connection: subclass of INSPECTIONS where
                         Order-for-Tanker = 1

DANGEROUS-CAPTAINS
  interclass connection: subclass of OFFICERS where is a value of
                         Involved-Captain of INCIDENTS

BANNED-SHIPS
  description: ships banned from U.S.coastal waters
  interclass connection: subclass of SHIPS where specified
  member attributes:
     Date-Banned
        valueclass: DATES

OILTANKERS-REQUIRING-INSPECTION
  interclass connection: subclass of OILTANKERS where specified

BANNED-OILTANKERS
  interclass connection: subclass of SHIPS where is in
```

```
                          BANNED-SHIPS and is in OILTANKERS

SAFE-SHIPS
  description: ships that are considered good risks
  interclass connection: subclass of SHIPS where is not
                         in BANNED-SHIPS

SHIPS-TO-BE-MONITORED
  description: ships that are considered bad risks
  interclass connection: subclass of SHIPS where is in
                         BANNED-SHIPS
                        or is in OILTANKERS-REQUIRING-INSPECTION

SHIPTYPES
  description: types of ships
  interclass connection: grouping of SHIPS on common value of Type
  member attributes:
     Instances
        description: the instances of the type of ship
        valueclass: SHIPS
        derivation: same as Contents
        multi-valued
     Number-of-Ships-of-this-Type
        valueclass: INTEGERS
        derivation: number of members in Contents

CARGOTYPE-GROUPS
  interclass connection: grouping of SHIPS on common value
                         of Cargo-Types

TYPES-OF-HAZARDOUS-SHIPS
  interclass connection: grouping of SHIPS consisting of classes
                         BANNED-SHIPS, BANNED-OILTANKERS,
                         SHIPS-TO-BE-MONITORED

CONVOYS
  interclass connection: grouping of SHIPS as specified
  member attributes:
     Oiltanker-Constituents
        description: the oiltankers that are in the convoy
                     (if any)
        valueclass: SHIPS
        derivation: subvalue of Contents where is in
                    OILTANKERS
        multi-valued


CARGO-TYPE-NAMES
  description: the types of cargo
  interclass connection: subclass of STRINGS
MERCHANT-CARGO-TYPE-NAMES
  interclass connection: subclass of CARGO-TYPE-NAMES where
                         specified
COUNTRY-NAMES
  interclass connection: subclass of STRINGS where specified
SERIAL-NUMBERS
  interclass connection: subclass of STRINGS where format is
     "H"
     number where integer and >= 1 and <= 999
```

```
    "_"
    number where integer and >= 0 and <= 999999
DATES
  description: calendar dates in the range "1/1/75"
               to "31/12/79"
  interclass connection: subclass of STRINGS where format is
      day: number where >= 1 and <= 31
      "/"
      month: number where >= 1 and <= 12
      "/"
      year: number where >= 1970 and <= 2000
      where (if(month = 4 or = 5 or = 9 or = 11)
                then day <= 30)
             and if (month = 2 then day <= 29))
      ordering by year, month, day
ENGINE-TYPE-NAMES
  interclass connection: subclass of STRINGS where specified
GALLONS
  interclass connection: subclass of STRINGS where format is
                         number
      where integer
HULL-NUMBERS
  interclass connection: subclass of STRINGS where format is
                         number
      where integer
HULL-TYPE-NAMES
  description: single or double
  interclass connection: subclass of STRINGS where specified
INCIDENT-DESCRIPTIONS
  description: textual description of an accident
  interclass connection: subclass of STRINGS
KNOTS
  interclass connection: subclass of STRINGS where format is
      number where integer
MILES-PER-HOUR
  interclass connection: subclass of STRINGS where format is
      number where integer
PORT-NAMES
  interclass connection: subclass of STRINGS
PERSON-NAMES
  interclass connection: subclass of STRINGS
SHIP-NAMES
  interclass connection: subclass of STRINGS
SHIP-TYPE-NAMES
  description: the names of the ship types, for example
               merchant
  interclass connection: subclass of STRINGS where specified
```

15. APPENDIX E. TERMINAL SESSIONS WITH ADD.

Below are given two examples of terminal sessions on the ADD system.
The first shows how a beginner might re-act with the system, the latter shows how an experienced user can shorten the process of model
specification.
User responses are preceded by -> to distinguish them from program displays.

```
WELCOME. IS THIS THE FIRST TIME YOU
ARE USING THE A.D.D. SYSTEM?
WHENEVER I ASK A QUESTION LIKE THIS,
YOU MUST EITHER TYPE
YES
OR TYPE
NO
(YOU MUST ALWAYS PRESS "RETURN"
WHEN YOU ARE READY FOR ME TO
REPLY.)

->


DO YOU WISH TO START A NEW SPECIFICATION
(IF YOUR PREVIOUS TIME WAS UNSUCCESSFUL)?

->Y


THIS MEANS WE ARE STARTING A COMPLETELY
NEW SPECIFICATION,NO DATA EXISTS FROM A
PREVIOUS SESSION. LET'S BE SURE:
ARE YOU QUITE HAPPY ABOUT THIS?

->Y


A.D.D. IS AN AUTOMATIC DATABASE
DESIGN SYSTEM. TO GUIDE THE DESIGN,
YOU WILL BE GIVING LISTS OF:
   OBJECTS
   PROPERTIES
   SUBTYPES
 & GROUPINGS.
 BEFORE YOU GIVE THESE LISTS,
 IT WILL BE EXPLAINED EXACTLY WHAT
THESE ARE, USING A SCHOOL EXAMPLE.
 IF AT ANY STAGE YOU ARE IN ANY WAY
 UNSURE OF WHAT TO DO, TYPE
 ?
```

```
 AND YOU WILL GET SOME EXPLANATION
 OF WHAT IS REQUIRED. (THESE MUST
 BE TYPED ON A NEW LINE, AS SHOWN ABOVE.)
 WHEN YOU HAVE FINISHED READING THIS,
 PRESS "RETURN" FOR MORE INFORMATION.

->


 WHEN USING THIS PROGRAM THERE ARE
 ONLY 3 THINGS YOU WILL NEED TO DO:
 1) ENTER INFORMATION WHEN ASKED TO DO SO
(THAT IS, THE 4 LISTS ABOVE,WITH SOME
 DESCRIPTION OF THE ITEMS IN EACH LIST):
 ONLY WORDS IN CAPITAL LETTERS
 ARE "READ" BY THIS PROGRAM, OTHER
 WORDS ARE IGNORED.
 2) ANSWER A YES/NO QUESTION, BY TYPING
YES
OR BY TYPING
NO
OR BY HITTING RETURN,WHICH IS THE SAME AS
NO
 3) CHOOSE FROM A LIST OF OPTIONS.
SUCH A LIST WILL APPEAR AFTER THIS
INTRODUCTION, ASKING WHAT TYPE OF
INFORMATION YOU WANT TO GIVE.
SEEING AS THIS IS YOUR FIRST TIME,
JUST CHOOSE
ANY OF THESE
(THE LAST OPTION IN THE LIST)
AND I WILL PICK ONE FOR YOU,
  UNTIL YOU KNOW ALL THE OPTIONS.
 WELL, I HOPE YOU ENJOY USING A.D.D.

->

WHICH WOULD YOU LIKE TO DO:
  ENTER INFORMATION
  REVIEW INFORMATION
   CHANGE INFORMATION
  TERMINATE
  ANY OF THESE        ?

->A

I SHALL CHOOSE FOR YOU.
IF YOU DON'T LIKE MY CHOICE, TYPE
QUIT
(ON A NEW LINE)

PLEASE LIST THE OBJECTS.
DO YOU WANT TO KNOW HOW TO PROCEED?

->Y

 AN OBJECT IS SOMETHING THAT EXISTS
IN YOUR ORGANISATION, SOMETHING
CONCRETE (TEACHER,ROOM) OR ABSTRACT(MARK),
```

```
THAT IS IMPORTANT IN THE ENVIRONMENT
FOR WHICH THIS DATABASE IS BEING DESIGNED.
EVENTS ARE ALSO OBJECTS, EG. IN A
SCHOOL ENVIRONMENT,
ENROLLING
IS A RELEVANT OBJECT, INVOLVING
A PUPIL AND THE SUBJECTS HE TAKES.
WHEN THE DATABASE EXISTS, THE QUESTIONS
YOU WILL ASK THE COMPUTER, WILL BE ABOUT
THESE OBJECTS, EG
"LIST ALL TEACHERS"
"WHAT IS THE HIGHEST MARK?"
ETC.DO YOU WANT TO KNOW MORE ABOUT THIS?

->NO


NOW, PLEASE GIVE AS MANY RELEVANT
OBJECTS AS POSSIBLE, THEN TYPE
QUIT

->there are STUDENTS taking COURSES  given by LECTURERS

also MARKS

->QUIT
WOULD YOU LIKE TO GIVE DETAILS ON MARKS
REPLY YES/NO OR "QUIT"

->N

->WOULD YOU LIKE TO GIVE DETAILS ON LECTURERS
REPLY YES/NO OR "QUIT"

->Y

WHAT DOES THIS MEAN EXACTLY? GIVE
A BRIEF STATEMENT ,ENDING WITH
!
OR IF IT IS OBVIOUS, JUST TYPE
!

->?


THE MEANING IS SOMETHING TO TELL
EXACTLY WHAT THIS OBJECT IS.
EG. FOR STUDENT, MEANING COULD BE:
ANYONE REGISTERED AT UCT THIS YEAR,
PART- OR FULL-TIME!

->ACADEMIC STAFF!

GIVE THE (LIST OF) PROPERTIES
TO IDENTIFY SUCH AN OBJECT UNIQUELY

GIVE PROPERTY NAME

->NAME
```

```
ERROR.NO SUCH ATTRIBUTE
AT PRESENT. WOULD YOU LIKE TO RESUBMIT?
REPLY "YES" OR "NO"

->NO

ANY MORE PROPERTIES?

->NO

 IS THERE ANOTHER WAY TO IDENTIFY THE LECTURERS

->NO



CAN TWO OF THESE OBJECTS
ALWAYS BE DISTINGUISHED FROM EACH OTHER?

->?

IF DUPLICATES ARE ALLOWED FOR AN OBJECT,
IT MEANS MORE THAN ONE SUCH OBJECT
CAN HAVE THE EXACT SAME IDENTIFIER.
THAT IS, THERE CAN BE 2 SUCH OBJECTS
HAVING ALL THEIR PROPERTIES IDENTICAL.
EG. DUPLICATES SHOULD NOT BE ALLOWED
FOR STUDENTS
   BUT DUPLICATES ARE ALLOWED FOR MARKS.

->X

ERROR. ONLY ANSWER "YES" OR "NO"

->NO

WOULD YOU LIKE TO GIVE DETAILS ON COURSES
REPLY YES/NO OR "QUIT"

->QUIT

            1
WHICH WOULD YOU LIKE TO DO:
  ENTER INFORMATION
  REVIEW INFORMATION
   CHANGE INFORMATION
  TERMINATE
  ANY OF THESE        ?

->A

I SHALL CHOOSE FOR YOU.
IF YOU DON'T LIKE MY CHOICE, TYPE
QUIT
(ON A NEW LINE)

PLEASE LIST THE PROPERTIES.
DO YOU WANT TO KNOW HOW TO PROCEED?
```

```
->Y
A PROPERTY IS SOMETHING
THAT DESCRIBES AN OBJECT,
SUBTYPE OR GROUPING; EG.
NAME, AGE, SIZE ETC.
DO YOU WANT TO KNOW MORE ABOUT THIS?

->UES
ERROR. ONLY ANSWER "YES" OR "NO"

->YES


 A PROPERTY IS A QUALITY THAT
 AN OBJECT POSSESSES. FOR EXAMPLE,
 THE OBJECT PUPIL HAS PROPERTIES
 NAME, AGE, SUBJECT, MARKS-GAINED,ETC.
 WHEN YOU ASK FOR DATA FROM THE COMPUTER
 ABOUT AN OBJECT, IT IS THE VALUES
 OF ITS PROPERTIES THAT YOU ARE WANTING.
 PROPERTIES ARE USUALLY NUMBER(S) OR
 WORD(S), BUT OFTEN THE PROPERTY OF
 ONE OBJECT IS ITSELF AN OBJECT IN THE
 DATABASE.
 E.G. MARKS-GAINED HAS AS ITS VALUE(S)
SOME OF THE MARK OBJECT(S) IN THE DATABASE.


NOW, GIVE AS MANY RELEVANT
PROPERTIES AS POSSIBLE, THEN TYPE
QUIT

-> NAME MARKGAINED AGE


->QUIT
WOULD YOU LIKE TO GIVE DETAILS ON AGE
REPLY YES/NO OR "QUIT"

->Y

IS THIS A PROPERTY OF THE DATABASE AS A WHOLE,
(AND NOT A PROPERTY OF INDIVIUAL OBJECTS)?

->?

AS AN EXAMPLE, NUMBER-OF-PUPILS IS NOT
 A PROPERTY OF AN INDIVIDUAL PUPIL; IT
IS A PROPERTY OF THE DATABASE ITSELF,VIZ:
HOW MANY PUPILS ARE STORED ON THE DATABASE.
OTHER EXAMPLES:AVERAGE-AGE,TOTAL-FEES.
HENCE THIS PROPERTY APPLIES TO
THE DATABASE AS A WHOLE, WHILE OTHERS
LIKE MARK-GAINED APPLY
TO INDIVIDUAL OBJECTS.
PROPERTIES OF THE DATABASE
AS A WHOLE ARE ONES THAT
WILL ONLY HAVE ONE VALUE,
AT ALL TIMES.
```

```
THIS IS DIFFERENT FROM
ORDINARY PROPERTIES, WHICH
APPLY TO SEVERAL OBJECTS, EG.
NAME OCCURS SEVERAL TIMES,
ONCE FOR EVERY PUPIL IN THE
DATABASE.SO, NOW TELL ME:
IS THIS PROPERTY APPLICABLE
TO THE DATABASE AS A WHOLE?

->DONT THINK SO

ERROR. ONLY ANSWER "YES" OR "NO"

->NO


WHAT IS THE MEANING OF THIS PROPERTY?
GIVE A BRIEF DESCRIPTION,
ENDING WITH AN EXCLAMATION MARK:
!
IF IT IS OBVIOUS,JUST TYPE
!

->!

WHAT OBJECTS HAVE THIS PROPERTY?
GIVE THE OBJECT NAME PLEASE.
GIVE NAME

->?

THIS TELLS WHAT OBJECTS
HAVE THIS PROPERTY. EG. MAXIMUM-MARK:
THIS IS A PROPERTY OF THE OBJECT EXAM.
WHAT IS WANTED IS: WHAT
OBJECTS/SUBTYPES/GROUPINGS
HAVE THIS PROPERTY. E.G
PROPERTY NAME COULD BE A
PROPERY OF SUBTYPE TEACHER,
OF SUBTYPE PUPIL, OR OF
OBJECT SUBJECT. SO PLEASE
TELL ME WHICH PARTICULAR OBJECT
THIS PROPERTY BELONGS TO

->STUDNTS
NO SUCH THING EXISTS YET
IS YOUR SPELLING CORRECT? "YES"/"NO"

->NO

GIVE NAME

->STUDENTS


WHAT KIND OF VALUE(S)
WILL THIS PROPERTY TAKE ON:
COULD BE REAL/BOOLEAN/STRINGS/INTEGER,
```

```
OR COULD BE SOME (OTHER) OBJECT
IN THE DATABASE.

->?

VALUE TYPE: WHAT KIND OF VALUES
 THIS PROPERTY HAS. A VALUE CAN BE A
 INTEGER (EG. 1  3  27)
 OR REAL(NUMBER WITH A FRACTION) OR
 STRING (ANY CHARACTERS EG SMITH, R2D2)
 BOOLEAN ("YES" OR "NO"; NOTHING ELSE)
 OR IT CAN BE SOME (OTHER) OBJECT
 IN THE DATABASE
(MANY PROPERTIES HAVE
AS THEIR VALUE, OTHER OBJECTS
OR SUBTYPES IN THE DATABASE:
SUCH AS SUBJECTS-TAKEN,
REGISTER-CLASS-TEACHER, ETC.)

->INTEGER


HOW MANY VALUES WILL THE PROPERTY HAVE?
ANSWER 1, IF EACH OBJECT HAS NO MORE
          THAN ONE VALUE FOR THIS PROPERTY
       M, IF MANY VALUES(ANY NUMBER OF VALUES
          IS POSSIBLE)
       V, IF VARIES BETWEEN 2 LIMITS,
          EG. FROM 1 TO 3 VALUES

->1

CAN THIS PROPERTY HAVE NO VALUE AT ALL,
IN SOME CIRCUMSTANCES?

->?

A PROPERTY MAY HAVE
NO VALUE AT ALL:
MEANS IT CAN BE BLANK.
EXAMPLE: PHONE-NUMBER.
IF SUBJECT-TAUGHT IS A PROPERTY OF
TEACHER, IT CANNOT BE BLANK.
CAN THIS PROPERTY BE BLANK?
THIS MEANS YOU CAN NEVER SAY SOMEONE
IS A TEACHER, UNLESS YOU SAY WHAT
SUBJECT HE TEACHES.

->NO


IS IT REASONABLE TO ASSUME THAT THIS
PROPERTY IS ALLOWED TO CHANGE ITS
VALUE ,IN TIME?
(AN EXCEPTION: BIRTHDATE)

->YES
```

```
CAN THIS PROPERTY HAVE
THE SAME VALUE
AT MORE THAN 1 PLACE IN THE DATABASE?

->?

A PROPERTY IS UNIQUE IF,
THROUGHOUT THE WHOLE DATABASE,
THIS PROPERTY NEVER HAS THE SAME
VALUE IN MORE THAN ONE PLACE;
OTHERWISE, IT CAN HAVE THE
SAME VALUE AT MORE THAN 1
PLACE IN THE DATABASE.
(EG. IF ROOM-NUMBER IS A PROPERTY OF
CLASSROOM, IT IS UNIQUE,
NO MORE THAN ONE CLASSROOM
CAN HAVE A SPECIFIC ROOM-NUMBER.
CAN THIS PROPERTY HAVE THE SAME
VALUE AT MORE THAN ONE PLACE
IN THE DATABASE?

->YES

DOES THIS PROPERTY
EXHAUST ITS VALUE CLASS?

->?

EXHAUSTING VALUE CLASS:
EXAMPLE: IF SUBJECT-TAUGHT IS
A PROPERTY OF TEACHERS,
THEN SUBJECT-TAUGHT EXHAUSTS ITS
VALUE CLASS SINCE EVERY SUBJECT MUST BE
TAUGHT BY SOMEONE! I.E. EVERY SUBJECT
MUST BE THE SUBJECT-TAUGHT BY SOME TEACHER.
DOES THIS PROPERTY
EXHAUST ITS VALUECLASS?

->NO


IS THE INVERSE PART OF THE DATABASE?

->?

AN INVERSE IS ONE OF A PAIR
OF REFERENCES TO THE SAME THING.
EXMAPLE: SUBJECTS-TAUGHT(PROPERTY OF
TEACHER) AND TEACHERS-HEREOF
(PROPERTY OF SUBJECT) ARE INVERSES.
IS THE INVERSE OF THIS
PROPERTY IMPORTANT?

->NO
```

```
WHAT IS ITS DERIVATION?:
 NONE
 SAME
 EXPRESSION
 IF IN SUBTYPE
 MATCH
 OTHER              ?

->?

>>>DERIVATIONS: USUALLY NONE!
BUT IN SPECIAL CASES, IT COULD BE THAT
THE PROPERTY YOU ARE CONSIDERING IS:
 - THE SAME AS ANOTHER PROPERTY
 - IF THE OBJECT IS IN A PARTICULAR
SUBTYPE OR NOT
 - IF IT MATCHES TO ANOTHER PROPERTY
 - EXPRESSION: ARITHMETIC EXPRESSION
INVOLVING OTHER PROPERTIES, EG
MARK1 + MARK2 / 2
 - OTHER: THIS IS IF ITS DERIVATION
IS ONE OF THE OTHER DERIVATIVES
(YOU PROBABLY DON'T KNOW ABOUT
 THESE YET!)

->N

WOULD YOU LIKE TO GIVE DETAILS ON MARKGAINED
REPLY YES/NO OR "QUIT"

->QUIT

WHICH WOULD YOU LIKE TO DO:
  ENTER INFORMATION
  REVIEW INFORMATION
   CHANGE INFORMATION
  TERMINATE
  ANY OF THESE        ?

->R
DO YOU WANT TO KNOW HOW TO GO ABOUT
REVIEWING THINGS?

->YES


REVIEWING LETS YOU LOOK AT
INFORMATION ALREADY GIVEN.
GIVE THE NAMES OF THE THINGS
THAT YOU WANT TO SEE
ONE AT A TIME.
DO YOU WANT MORE INFORMATION?

->NO


WHAT ARE YOU CONCERNED WITH:
(ENTER ONE AT A TIME PLEASE)
```

```
->LECTURERS

OBJECT: LECTURERS
MEANING: ACADEMIC STAFF

DUPLICATES ALLOWED
IDENTIFIABLE BY:
          1)
 NAME
--------------------


WOULD YOU LIKE TO REVIEW ANYTHING ELSE?

->YES

->AGE
PROPERTY NAME: AGE
CANNOT BE BLANK
NO OVERLAP ALLOWED
EXHAUSTS VALUE CLASS
NO. OF VALUES:              1
A PROPERTY OF OBJECT: STUDENTS
WHICH HAS VALUES THAT ARE INTEGER


WOULD YOU LIKE TO REVIEW ANYTHING ELSE?

->NO


WHICH WOULD YOU LIKE TO DO:
  ENTER INFORMATION
  REVIEW INFORMATION
   CHANGE INFORMATION
  TERMINATE
  ANY OF THESE        ?

->T

ARE YOU
  LEAVING
  FINISHING
  DROPPING THIS RUN
  ANY OF THESE         ?

->?

LEAVING IS WHAT YOU DO IF
 YOU HAVE NOT COMPLETED YOUR
DATABASE SPECIFICATION ALTOGETHER,
BUT WISH TO LEAVE OFF FOR A WHILE,
TO CONTINUE WITH IT SOME OTHER TIME.
FINISHING IS WHAT HAPPENS WHEN YOU
ARE QUITE CERTAIN THAT YOU HAVE
ABSOLUTELY NO MORE INFORMATION TO GIVE
AND YOU WISH TO END THE SPECIFICATION
(FOREVER)
```

```
DROPPING A RUN MEANS THAT EVERY THING
YOU ENTERED DURING THIS PARTICULAR
RUN OF THE PROGRAM WILL BE ABANDONED.
DO THIS IF YOU WANT TO STOP BECAUSE
YOU HAVE MADE TOO MANY BAD ERRORS.
ANY OF THESE - IF YOU DON'T
WANT TO CHOOSE FROM THE ABOVE,
TYPE THIS.

->LLEAVING


I WOULD LIKE TO BE SURE I HAVE AS
COMPLETE A DESCRIPTION AS
POSSIBLE OF ALL
THIS NEW INFORMATION.
CAN WE GO THROUGH AND FILL IN DETAILS
WHERE NECESSARY ?

->NO I HAVE TO RUN!
```

```
WELCOME. IS THIS THE FIRST TIME YOU
ARE USING THE A.D.D. SYSTEM?
WHENEVER I ASK A QUESTION LIKE THIS,
YOU MUST EITHER TYPE
YES
OR TYPE
NO
(YOU MUST ALWAYS PRESS "RETURN"
WHEN YOU ARE READY FOR ME TO
REPLY.)

->Y


THIS MEANS WE ARE STARTING A COMPLETELY
NEW SPECIFICATION,NO DATA EXISTS FROM A
PREVIOUS SESSION. LET'S BE SURE:
ARE YOU QUITE HAPPY ABOUT THIS?

->Y


A.D.D. IS AN AUTOMATIC DATABASE
DESIGN SYSTEM. TO GUIDE THE DESIGN,
YOU WILL BE GIVING LISTS OF:
   OBJECTS
   PROPERTIES
   SUBTYPES
 & GROUPINGS.
 BEFORE YOU GIVE THESE LISTS,
 IT WILL BE EXPLAINED EXACTLY WHAT
THESE ARE, USING A SCHOOL EXAMPLE.
 IF AT ANY STAGE YOU ARE IN ANY WAY
 UNSURE OF WHAT TO DO, TYPE
 ?
 AND YOU WILL GET SOME EXPLANATION
 OF WHAT IS REQUIRED. (THESE MUST
 BE TYPED ON A NEW LINE, AS SHOWN ABOVE.)
 WHEN YOU HAVE FINISHED READING THIS,
 PRESS "RETURN" FOR MORE INFORMATION.

->

 WHEN USING THIS PROGRAM THERE ARE
 ONLY 3 THINGS YOU WILL NEED TO DO:
 1) ENTER INFORMATION WHEN ASKED TO DO SO
(THAT IS, THE 4 LISTS ABOVE,WITH SOME
 DESCRIPTION OF THE ITEMS IN EACH LIST):
 ONLY WORDS IN CAPITAL LETTERS
 ARE "READ" BY THIS PROGRAM, OTHER
 WORDS ARE IGNORED.
 2) ANSWER A YES/NO QUESTION, BY TYPING
YES
OR BY TYPING
NO
OR BY HITTING RETURN,WHICH IS THE SAME AS
```

```
NO
 3) CHOOSE FROM A LIST OF OPTIONS.
SUCH A LIST WILL APPEAR AFTER THIS
INTRODUCTION, ASKING WHAT TYPE OF
INFORMATION YOU WANT TO GIVE.
SEEING AS THIS IS YOUR FIRST TIME,
JUST CHOOSE
ANY OF THESE
(THE LAST OPTION IN THE LIST)
AND I WILL PICK ONE FOR YOU,
  UNTIL YOU KNOW ALL THE OPTIONS.
 WELL, I HOPE YOU ENJOY USING A.D.D.

->


WHICH WOULD YOU LIKE TO DO:
  ENTER INFORMATION
  REVIEW INFORMATION
   CHANGE INFORMATION
  TERMINATE
  ANY OF THESE        ?

->O

DO YOU WANT TO KNOW HOW TO PROCEED?

->N

NOW, PLEASE GIVE AS MANY RELEVANT
OBJECTS AS POSSIBLE, THEN TYPE
QUIT

->SHIPS INSPECTIONS COUNTRIES ASSIGNMENTS INCIDENTS OFFICERS
ENGINES


->Q

WOULD YOU LIKE TO GIVE DETAILS ON ENGINES
REPLY YES/NO OR "QUIT"

->Q


WHICH WOULD YOU LIKE TO DO:
  ENTER INFORMATION
  REVIEW INFORMATION
   CHANGE INFORMATION
  TERMINATE
  ANY OF THESE        ?

->S

DO YOU WANT TO KNOW HOW TO PROCEED?

->N
```

```
NOW PLEASE GIVE AS MANY
RELEVANTS SUBTYPES AS POSSIBLE,THEN TYPE
QUIT

->OILTANKERS RURITANIANSHIPS RURITANIANOILTANKERS MERCHANTSHIPS
BANNEDSHIPS


->OILSPILLS SAFESHIPS MOSTRECENTINSPECTIONS DANGEROUSCAPTAINS


->OILTANKERSREQUIRINGINSPECTION SHIPSTOBEMONITORED
BANNEDOILTANKERS


->Q GDO YOU WANT TO KNOW HOW TO PROCEED?

->N

NOW PLEASE GIVE AS MANY
RELEVANT GROUPINGS AS POSSIBLE,THEN TYPE
QUIT

->SHIPTYPEGROUPS CARGOTYPEGROUPS TYPESOFHAZARDOUSSHIPS CONVOYS

->QUIT P


DO YOU WANT TO KNOW HOW TO PROCEED?

->N

NOW, GIVE AS MANY RELEVANT
PROPERTIES AS POSSIBLE, THEN TYPE
QUIT

->SHIPS SHIPNAME HULLNUMBER TYPE COUNTRYOFREGISTRY NAMEOFHOMEPORT
CARGOTYPES


->CAPTAIN SHIPENGINES INCIDENTSINVOLVEDIN INSPECTIONS
TANKERINSPECTED


->INSPECTIONDATE ORDERFORTANKER NUMBEROFINSPECTIONS COUNTRIES
COUNTRYNAME


->SHIPSREGISTEREDHERE OFFICERS OFFICERNAME COUNTRYOFLICENCE
DATECOMMISSIONED


->SENIORITY COMMANDER SUPERIORS SUBORDINATES CONTACTS ENGINES
SERIALNUMBER
```

```
->KINDOFENGINE INCIDENTS INVOLVEDSHIP INCIDENTDATE DESCRIPTION
INVOLVEDCAPTAIN

->ASSIGNMENTS SHIP OFFICER OILTANKERS HULLTYPE ISTANKERBANNED

->INSPECTIONSOFTHISTANKER NUMBEROFTIMESINSPECTED LASTINSPECTION


->LASTTWOINSPECTIONS DATELASTEXAMINED OILSPILLSINVOLVEDIN
ABSOLUTETOPLEGALSPEED


->TOPLEGALSPEEDINMILESPERHOUR OILSPILLS AMOUNTSPILLED SEVERITY
TOTALSPILLED


->BANNEDSHIPS DATEBANNED SHIPTYPEGROUPS INSTANCES
NUMBEROFSHIPSOFTHISTYPE

->CONVOYS OILTANKERCONSTITUENTS


->Q L


I WOULD LIKE TO BE SURE I HAVE AS
COMPLETE A DESCRIPTION AS
POSSIBLE OF ALL
THIS NEW INFORMATION.
CAN WE GO THROUGH AND FILL IN DETAILS
WHERE NECESSARY ?

->Y


AT ANY TIME, IF YOU WANT TO GO,
YOU CAN DO SO BY TYPING
QUIT
TYPE "QUIT" TO GO,ELSE HIT "RETURN"

->N

WOULD YOU LIKE TO GIVE DETAILS ON ENGINES
REPLY YES/NO OR "QUIT"

->Y

WHAT DOES THIS MEAN EXACTLY? GIVE
A BRIEF STATEMENT ,ENDING WITH
!
```

```
OR IF IT IS OBVIOUS, JUST TYPE
!

->SHIP ENGINES!

GIVE THE (LIST OF) PROPERTIES
TO IDENTIFY SUCH AN OBJECT UNIQUELY

GIVE PROPERTY NAME

->SERIALNUMBER
ANY MORE PROPERTIES?

->N

 IS THERE ANOTHER WAY TO IDENTIFY THE ENGINES

->N


CAN TWO OF THESE OBJECTS
ALWAYS BE DISTINGUISHED FROM EACH OTHER?

->Y

WOULD YOU LIKE TO GIVE DETAILS ON OFFICERS
REPLY YES/NO OR "QUIT"

->Y

WHAT DOES THIS MEAN EXACTLY? GIVE
A BRIEF STATEMENT ,ENDING WITH
!
OR IF IT IS OBVIOUS, JUST TYPE
!

->ALL CERTIFIED OFFICERS OF SHIPS!


GIVE THE (LIST OF) PROPERTIES
TO IDENTIFY SUCH AN OBJECT UNIQUELY

GIVE PROPERTY NAME

->OFFICERNAME
ANY MORE PROPERTIES?

->N

 IS THERE ANOTHER WAY TO IDENTIFY THE OFFICERS

->N


CAN TWO OF THESE OBJECTS
ALWAYS BE DISTINGUISHED FROM EACH OTHER?
```

```
->Y

WOULD YOU LIKE TO GIVE DETAILS ON INCIDENTS
REPLY YES/NO OR "QUIT"

->Y


..... This process continues ...


I NEED DATA ON SOME GROUPINGS
TYPE "QUIT" TO GO,ELSE HIT "RETURN"

->N

WOULD YOU LIKE TO GIVE DETAILS ON CONVOYS
REPLY YES/NO OR "QUIT"

->Y

THE MEANING OF A GROUPING IS
A CONCISE STATEMENT OF WHAT
THAT GROUPING REFERS TO.
ENTER THE MEANING, ENDING WITH A
!
OR, IF IT IS OBVIOUS, JUST TYPE
!

->!

WHAT IS THE DEFINITION
WHEREBY WE GROUP THE OBJECTS:
 SPECIFIABLE BY USER
 COMMON VALUE OF SOME PROPERTY
 LIST OF OBJECT TYPES   ?

->S

WHAT KIND OF OBJECT OR SUBTYPE WILL BE
GROUPED ?
GIVE NAME

->SHIPS

WOULD YOU LIKE TO GIVE DETAILS ON TYPESOFHAZARDOUSSHIPS
REPLY YES/NO OR "QUIT"

->Y

THE MEANING OF A GROUPING IS
A CONCISE STATEMENT OF WHAT
THAT GROUPING REFERS TO.
ENTER THE MEANING, ENDING WITH A
!
OR, IF IT IS OBVIOUS, JUST TYPE
!
```

```
->!

WHAT IS THE DEFINITION
WHEREBY WE GROUP THE OBJECTS:
 SPECIFIABLE BY USER
 COMMON VALUE OF SOME PROPERTY
 LIST OF OBJECT TYPES   ?

->L

WHAT KIND OF OBJECT OR SUBTYPE WILL BE
GROUPED?
GIVE NAME

->SHIPS

WHAT OBJECT TYPES ARE THERE
IN THE LIST?
ENTER THESE ONE BY ONE.


GIVE NAME

->BANNEDSHIPS
ANY MORE OBJECTS?

->Y

GIVE NAME

->BANNEDOILTANKERS
ANY MORE OBJECTS?

->Y

GIVE NAME

->SHIPSTOBEMONITORED

ANY MORE OBJECTS?

->N

WOULD YOU LIKE TO GIVE DETAILS ON CARGOTYPEGROUPS
REPLY YES/NO OR "QUIT"

->Y

THE MEANING OF A GROUPING IS
A CONCISE STATEMENT OF WHAT
THAT GROUPING REFERS TO.
ENTER THE MEANING, ENDING WITH A
!
OR, IF IT IS OBVIOUS, JUST TYPE
!

->!

WHAT IS THE DEFINITION
```

```
WHEREBY WE GROUP THE OBJECTS:
 SPECIFIABLE BY USER
 COMMON VALUE OF SOME PROPERTY
 LIST OF OBJECT TYPES   ?

->C

WHAT KIND OF OBJECT OR SUBTYPE WILL BE
GROUPED ?
GIVE NAME

->SHIPS


WHICH PROPERTIES MUST BE USED
TO FORM THE GROUPS: ENTER ONE BY ONE

GIVE PROPERTY NAME

->CARGOTYPES

ANY MORE PROPERTIES?

->N

WOULD YOU LIKE TO GIVE DETAILS ON SHIPTYPEGROUPS
REPLY YES/NO OR "QUIT"

->Y

THE MEANING OF A GROUPING IS
A CONCISE STATEMENT OF WHAT
THAT GROUPING REFERS TO.
ENTER THE MEANING, ENDING WITH A
!
OR, IF IT IS OBVIOUS, JUST TYPE
!

->TYPES OF SHIPS!

WHAT IS THE DEFINITION
WHEREBY WE GROUP THE OBJECTS:
 SPECIFIABLE BY USER
 COMMON VALUE OF SOME PROPERTY
 LIST OF OBJECT TYPES   ?

->C

WHAT KIND OF OBJECT OR SUBTYPE WILL BE
GROUPED ?
GIVE NAME

->SHIPS


WHICH PROPERTY MUST BE USED
TO FORM THE GROUPS: ENTER ONE BY ONE
```

```
GIVE PROPERTY NAME

->TYPE
ANY MORE PROPERTIES?

->N

I NEED DATA ON SOME SUBTYPES
TYPE "QUIT" TO GO,ELSE HIT "RETURN"

->N

WOULD YOU LIKE TO GIVE DETAILS ON BANNEDOILTANKERS
REPLY YES/NO OR "QUIT"

->Q

I NEED DATA ON SOME VALUE TYPES
TYPE "QUIT" TO GO,ELSE HIT "RETURN"

->Q
```

16. APPENDIX F. RELATION SCHEME GENERATED BY ADD.

```
------------------------------
        RELATION 1
          TYPE 6
------------------------------
COUNTRYOFREGISTRY          : COUNTRIES#
SHIPSREGISTEREDHERE        : SHIPS#              *KEY*
------------------------------
        RELATION 2
          TYPE 2
------------------------------
CARGOTYPES                 : STRINGS             *KEY*
SHIPS#                     : INTEGER             *KEY*
------------------------------
        RELATION 3
          TYPE 2
------------------------------
SHIPENGINES                : ENGINES#            *KEY*
SHIPS#                     : INTEGER             *KEY*
------------------------------
        RELATION 4
          TYPE 5
------------------------------
INCIDENTSINVOLVEDIN        : INCIDENTS#          *KEY*
INVOLVEDSHIP               : SHIPS#
------------------------------
        RELATION 5
          TYPE 6
------------------------------
TANKERINSPECTED            : OILTANKERS#
INSPECTIONSOFTHISTANKER    : INSPECTIONS#        *KEY*
------------------------------
        RELATION 6
          TYPE 12
------------------------------
INSPECTIONS#               : INTEGER             *KEY*
ORDERFORTANKER             : INTEGER
------------------------------
        RELATION 7
          TYPE 12
------------------------------
OFFICERS#                  : INTEGER             *KEY*
SENIORITY                  : INTEGER
------------------------------
        RELATION 8
          TYPE 4
------------------------------
SUPERIORS                  : OFFICERS#           *KEY*
SUBORDINATES               : OFFICERS#           *KEY*
------------------------------
        RELATION 9
          TYPE 2
------------------------------
CONTACTS                   : OFFICERS#           *KEY*
OFFICERS#                  : INTEGER             *KEY*
------------------------------
        RELATION 10
```

```
          TYPE 2
-----------------------------
LASTTWOINSPECTIONS          : INSPECTIONS#              *KEY*
OILTANKERS#                 : INTEGER                   *KEY*
-----------------------------
         RELATION 11
           TYPE 3
-----------------------------
LASTINSPECTION              : MOSTRECENTINSPECTIONS#    *KEY*
DATELASTEXAMINED-YEAR       : INTEGER
DATELASTEXAMINED-MONTH      : INTEGER
DATELASTEXAMINED-DAY        : INTEGER
-----------------------------
         RELATION 12
           TYPE 2
-----------------------------
OILSPILLSINVOLVEDIN         : INCIDENTS#                *KEY*
OILTANKERS#                 : INTEGER                   *KEY*
-----------------------------
         RELATION 13
           TYPE 12
-----------------------------
QENTRY#                     : INTEGER                   *KEY*
TOPLEGALSPEEDINMILESPERHOUR : INTEGER
-----------------------------
         RELATION 14
           TYPE 12
-----------------------------
OILSPILLS#                  : INTEGER                   *KEY*
SEVERITY                    : INTEGER
-----------------------------
         RELATION 15
           TYPE 2
-----------------------------
INSTANCES                   : SHIPS#                    *KEY*
SHIPTYPEGROUPS#             : INTEGER                   *KEY*
-----------------------------
         RELATION 16
           TYPE 2
-----------------------------
OILTANKERCONSTITUENTS       : SHIPS#                    *KEY*
CONVOYS#                    : INTEGER                   *KEY*
-----------------------------
         RELATION 17
           TYPE 1
-----------------------------
SHIPS#                      : INTEGER                   *KEY*
HULLNUMBER                  : INTEGER
-----------------------------
         RELATION 18
           TYPE 1
-----------------------------
SHIPS#                      : INTEGER                   *KEY*
SHIPNAME                    : STRINGS
TYPE                        : STRINGS
NAMEOFHOMEPORT              : STRINGS
-----------------------------
         RELATION 19
           TYPE 1
```

```
-----------------------------
INSPECTIONS#                 : INTEGER          *KEY*
TANKERINSPECTED              : OILTANKERS#
INSPECTIONDATE-YEAR          : INTEGER
INSPECTIONDATE-MONTH         : INTEGER
INSPECTIONDATE-DAY           : INTEGER
-----------------------------
        RELATION 20
          TYPE 1
-----------------------------
QENTRY#                      : INTEGER          *KEY*
ENTRYKEY                     : ALPHABETIC
-----------------------------
        RELATION 21
          TYPE 1
-----------------------------
QENTRY#                      : INTEGER          *KEY*
NUMBEROFINSPECTIONS          : INTEGER
ABSOLUTETOPLEGALSPEED        : INTEGER
TOTALSPILLED                 : INTEGER
-----------------------------
        RELATION 22
          TYPE 1
-----------------------------
COUNTRIES#                   : INTEGER          *KEY*
COUNTRYNAME                  : STRINGS
-----------------------------
        RELATION 23
          TYPE 1
-----------------------------
OFFICERS#                    : INTEGER          *KEY*
OFFICERNAME                  : STRINGS
-----------------------------
        RELATION 24
          TYPE 1
-----------------------------
OFFICERS#                    : INTEGER          *KEY*
COUNTRYOFLICENCE             : COUNTRIES#
DATECOMMISSIONED-YEAR        : INTEGER
DATECOMMISSIONED-MONTH       : INTEGER
DATECOMMISSIONED-DAY         : INTEGER
COMMANDER                    : OFFICERS#
-----------------------------
        RELATION 25
          TYPE 1
-----------------------------
ENGINES#                     : INTEGER          *KEY*
SERIALNUMBER-H               : STRINGS
SERIALNUMBER-NUMONE          : INTEGER
SERIALNUMBER-HYPHEN          : STRINGS
SERIALNUMBER-NUMTWO          : INTEGER
-----------------------------
        RELATION 26
          TYPE 1
-----------------------------
ENGINES#                     : INTEGER          *KEY*
KINDOFENGINE                 : STRINGS
-----------------------------
        RELATION 27
```

TYPE 1

INCIDENTS#	:	INTEGER	*KEY*
INVOLVEDSHIP	:	SHIPS#	
INCIDENTDATE-YEAR	:	INTEGER	
INCIDENTDATE-MONTH	:	INTEGER	
INCIDENTDATE-DAY	:	INTEGER	
DESCRIPTION	:	STRINGS	

RELATION 28
TYPE 1

INCIDENTS#	:	INTEGER	*KEY*
INVOLVEDCAPTAIN	:	OFFICERS#	

RELATION 29
TYPE 7

ASSIGNMENTS#	:	INTEGER	*KEY*
SHIP	:	SHIPS#	

RELATION 30
TYPE 7

ASSIGNMENTS#	:	INTEGER	*KEY*
OFFICER	:	OFFICERS#	

RELATION 31
TYPE 1

OILTANKERS#	:	INTEGER	*KEY*
HULLTYPE	:	STRINGS	
ISTANKERBANNED	:	BOOLEAN	
NUMBEROFTIMESINSPECTED	:	INTEGER	
LASTINSPECTION	:	MOSTRECENTINSPECTIONS#	

RELATION 32
TYPE 1

OILSPILLS#	:	INTEGER	*KEY*
AMOUNTSPILLED	:	INTEGER	

RELATION 33
TYPE 1

BANNEDSHIPS#	:	INTEGER	*KEY*
DATEBANNED-YEAR	:	INTEGER	
DATEBANNED-MONTH	:	INTEGER	
DATEBANNED-DAY	:	INTEGER	

RELATION 34
TYPE 1

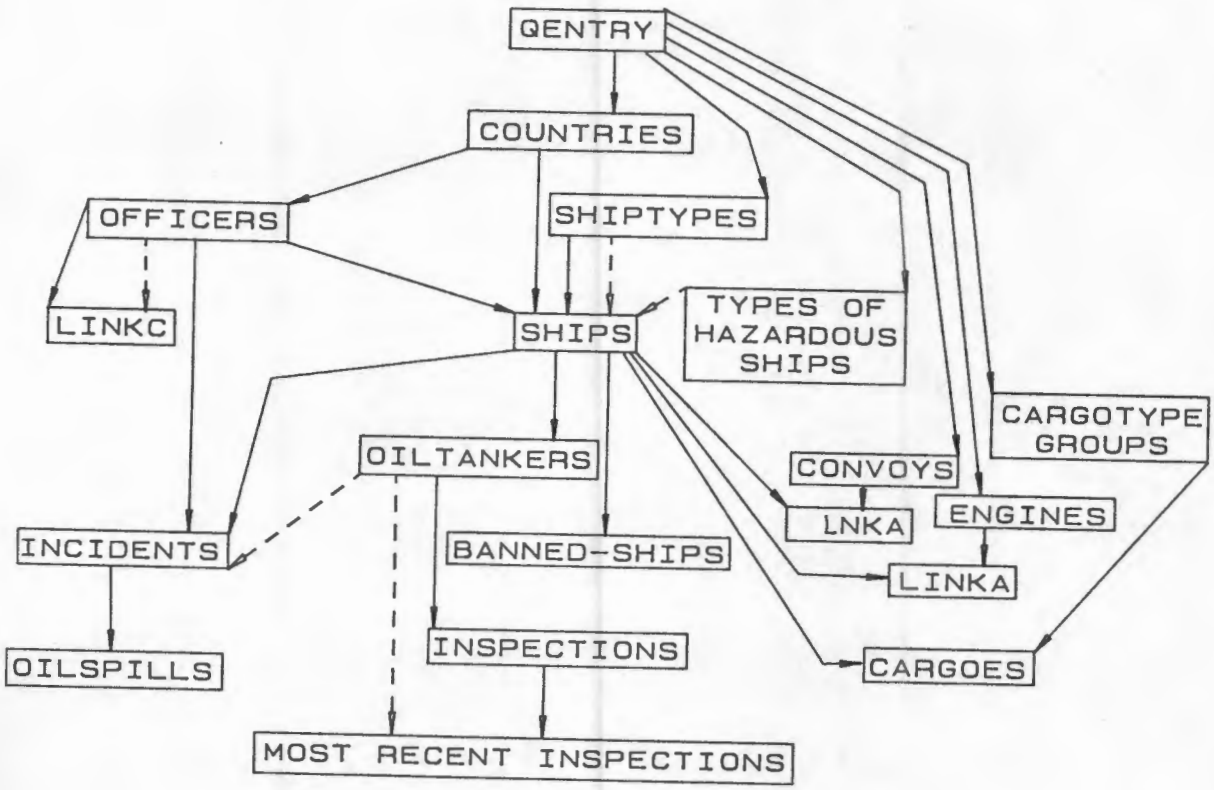
SHIPTYPEGROUPS#	:	INTEGER	*KEY*
CONTENTS	:	SHIPS#	

RELATION 35
TYPE 1

```
SHIPTYPEGROUPS#              : INTEGER          *KEY*
NUMBEROFSHIPSOFTHISTYPE      : INTEGER
-----------------------------
         RELATION 36
           TYPE 1
-----------------------------
CONVOYS#                     : INTEGER          *KEY*
CONTENTS                     : SHIPS#
-----------------------------
         RELATION 37
           TYPE 1
-----------------------------
CONVOYS#                     : INTEGER          *KEY*
CONTENTS                     : SHIPS#
-----------------------------
         RELATION 38
           TYPE 1
-----------------------------
CARGOTYPEGROUPS#             : INTEGER          *KEY*
CONTENTS                     : SHIPS#
-----------------------------
         RELATION 39
           TYPE 1
-----------------------------
TYPESOFHAZARDOUSSHIPS#       : INTEGER          *KEY*
CONTENTS                     : SHIPS#
-----------------------------
```

17. APPENDIX G. STRUCTURE OF THE NETWORK SYNTHESISED.

This diagram shows the structure of the network generated by ADD for the SDM model in Appendix D. For clarity, the names of the sets have been omitted; the "original" record names are given, not the 6-character versions; sets connecting QENTRY to another record for class attributes and "orderings", are not shown. These link QENTRY to OILTANKERS, OILSPILLS, OFFICERS and INSPECTIONS.



18. APPENDIX H. SCHEMA GENERATED BY ADD.

```
IDENTIFICATION DIVISION
    SCHEMA NAME IS QSYSTB   IN FILE QSYSTC
DATA DIVISION
DATA-NAME SECTION
    01   QENTAK
                   USAGE IS AREA-KEY
    01   QENTAN
                   USAGE IS AREA-NAME
AREA SECTION
AREA NAME IS     QAREAA
    MODE IS DATA
    ALLOCATE 100
                   PAGES
     10 OVERFLOW AT END
     1 OVERFLOW EVERY  9 DATA
    EXPANDABLE TO 300
    PAGES ARE 280 WORDS
    LOAD IS 75 PERCENT
RECORD SECTION
RECORD NAME IS BNNDSH

*
    LOCATION VIA SHIBNN   SET
    10 DTBNND
                   CHECK
                   PROCEDURE
                   DATEZH
    20 DTBN-D      PIC 9(     2)
                   USAGE IS COMP
    20 DTBN-M      PIC 9(     2)
                   USAGE IS COMP
    20 DTBN-Y      PIC 9(     4)
                   USAGE IS COMP
 RECORD NAME IS CONTRS
*COUNTRIES OF REGISTRY FOR SHIPS
    LOCATION VIA QSYSTD   SET
    10 CNTRYN      PIC X(    24)
 RECORD NAME IS CONVYS
    LOCATION VIA QSYSTE   SET
    10 CONVYU      PIC 9(     1)
                   USAGE IS COMP
 RECORD NAME IS CRGTYP
    LOCATION VIA CRGTYR   SET
    10 CRGTYQ      PIC X(    18)
*THE TYPE(S) OF CARGO THE SHIPS CAN CARRY
 RECORD NAME IS CRGTYS
    LOCATION VIA QSYSTF   SET
    10 CRGTYT      PIC X(    18)
*THE TYPE(S) OF CARGO THE SHIPS CAN CARRY
                   RESULT
                   PROCEDURE QEQUAL
                   SET CRGCRG
                   MEMBER CRGTYP
                   USING    CRGTYQ
                   CHECK
```

```
                    VALUE
                    NOT
                    NULL
     RECORD NAME IS ENGINS
    *    SHIP ENGINES
         LOCATION VIA QSYSTG   SET
         10 KNDFNG     PIC X(     3)
         10 SRLNMB
                       CHECK
                       PROCEDURE
                      ENGINU
         20 SRLN-H     PIC X(     9)
         20 SRLN-I     PIC X(     9)
         20 SRLN-N     PIC 9(     3)
                       USAGE IS COMP
         20 SRLN-O     PIC 9(     6)
                       USAGE IS COMP
     RECORD NAME IS LINKA
         LOCATION VIA SHPNGN   SET
         10 QSYSTH     PIC X(     1)
     RECORD NAME IS LINKC
         LOCATION VIA CNTCTS   SET
         RESERVE      1 POINTERS
         10 QSYSTI     PIC X(     1)
     RECORD NAME IS LNKA
         LOCATION VIA CONLNK   SET
         10 CONVYT     PIC 9(     1)
                       USAGE IS COMP
                       CHECK
                       VALUE
                       0
                       THRU 1
     RECORD NAME IS LSPLLT

*
         LOCATION VIA LSPLLV   SET
         10 MNTSPL     PIC 9(     6)
                       USAGE IS COMP
         10 SEVRTY     PIC 9(     6)
                       USAGE IS COMP
                       RESULT
                       PROCEDURE LNKJFC
                       ON THIS RECORD USING
                       MNTSPL
     RECORD NAME IS LTNKRS

*
         LOCATION VIA LTNKRU   SET
         RESERVE      1 POINTERS
         10 DTLSTX
                       CHECK
                       PROCEDURE
                       DATEZG
         20 DTLS-D     PIC 9(     2)
                       USAGE IS COMP
         20 DTLS-M     PIC 9(     2)
                       USAGE IS COMP
         20 DTLS-Y     PIC 9(     4)
                       USAGE IS COMP
```

```
        10 HLLTYP       PIC X(     6)
      *SPECIFICATION OF SINGLE OR DOUBLE HULL
        10 NMBRFT       PIC 9(     9)
                        USAGE IS COMP
                        RESULT
                        COUNT
                        SET NSPCTO
                        MEMBER NSPCTN
        10 OILTAQ       PIC 9(     1)
                        USAGE IS COMP
        10 RURITB       PIC 9(     1)
                        USAGE IS COMP
        10 STNKRB       PIC 9(     1)
                        USAGE IS COMP
       RECORD NAME IS MSTRCN

*
        LOCATION VIA NSPMST   SET
        10 QSYSTJ       PIC X(     1)
       RECORD NAME IS NCDNTS
      *ACCIDENTS INVOLVING SHIPS
        LOCATION VIA NCDNTT   SET
        10 DSCRPT       PIC X(   100)
      *TEXTUAL EXPLANATION OF THE ACCIDENT!
        10 LSPLLU       PIC 9(     1)
                        USAGE IS COMP
                        RESULT
                        PROCEDURE QEQUAL
                        ON THIS RECORD USING
                        OILSPI
                        CHECK
                        VALUE
                        0
                        THRU 1
        10 NCDNTD
                        CHECK
                        PROCEDURE
                        DATEZF
        20 NCDN-D       PIC 9(     2)
                        USAGE IS COMP
        20 NCDN-M       PIC 9(     2)
                        USAGE IS COMP
        20 NCDN-Y       PIC 9(     4)
                        USAGE IS COMP
        10 OILSPI       PIC 9(     1)
                        USAGE IS COMP
                        RESULT
                        PROCEDURE LNKF
                        ON THIS RECORD USING
                        DSCRPT
        10 OILSPJ       PIC 9(     1)
                        USAGE IS COMP
                        RESULT
                        SUM
                        ON THIS RECORD USING
                        OILSPI
                        OILSPK
                        CHECK
                        VALUE
```

```
                    NOT
                        1
      10 OILSPK     PIC 9(     1)
                    USAGE IS COMP
                    RESULT
                    COUNT
                    SET NCDLSP
                    MEMBER LSPLLT
                    CHECK
                    VALUE
                    MAX
                        1
 RECORD NAME IS NSPCTN
*INSPECTIONS OF OIL TANKERS
     LOCATION VIA NSPCTR   SET
      10 MOSTRE     PIC 9(     1)
                    USAGE IS COMP
                    RESULT
                    PROCEDURE LNKG
                    ON THIS RECORD USING
                    RDRFRT
      10 MOSTRF     PIC 9(     1)
                    USAGE IS COMP
                    RESULT
                    SUM
                    ON THIS RECORD USING
                    MOSTRE
                    MOSTRG
                    CHECK
                    VALUE
                    NOT
                        1
      10 MOSTRG     PIC 9(     1)
                    USAGE IS COMP
                    RESULT
                    COUNT
                    SET NSPMST
                    MEMBER MSTRCN
                    CHECK
                    VALUE
                    MAX
                        1
      10 NSPCTP
                    CHECK
                    PROCEDURE
                    DATEZD
      20 NSPC-D     PIC 9(     2)
                    USAGE IS COMP
      20 NSPC-M     PIC 9(     2)
                    USAGE IS COMP
      20 NSPC-Y     PIC 9(     4)
                    USAGE IS COMP
      10 NSPCTQ     PIC 9(     1)
                    USAGE IS COMP
                    CHECK
                    VALUE
                    0
                    THRU 1
 RECORD NAME IS OFFCRS
```

```
*ALL CERTIFIED OFFICERS OF SHIPS!
     LOCATION VIA LNKHFC   SET
     10 DANGER      PIC 9(     1)
                    USAGE IS COMP
                    RESULT
                    COUNT
                    SET NVLVDC
                    MEMBER NCDNTS
     10 DTCMMS
                    CHECK
                    PROCEDURE
                    DATEZE
     20 DTCM-D      PIC 9(     2)
                    USAGE IS COMP
     20 DTCM-M      PIC 9(     2)
                    USAGE IS COMP
     20 DTCM-Y      PIC 9(     4)
                    USAGE IS COMP
     10 FFCRNM      PIC X(    45)
 RECORD NAME IS QENTRY
     DIRECT QENTAK     QENTAN
     WITHIN QAREAA
     10 BSLTTP      PIC 9(     3)
                    USAGE IS COMP
     10 NMBRFN      PIC 9(     9)
*THE NUMBER OF INSPECTIONS IN THE DATABASE
                    USAGE IS COMP
                    RESULT
                    COUNT
                    SET NSPCTR
                    MEMBER NSPCTN
     10 TPLGLS      PIC 9(     4)
                    USAGE IS COMP
                    RESULT
                    PROCEDURE LNKIFC
                    ON THIS RECORD USING
                    BSLTTP
     10 TTLSPL      PIC 9(     6)
                    USAGE IS COMP
                    RESULT
                    SUM
                    SET LSPLLV
                    MEMBER LSPLLT
                    USING    MNTSPL
 RECORD NAME IS SHIPS
*ALL SHIPS WITH PROTENTIALLY HAZARDOUS CARGOES
     LOCATION VIA SHPSRG   SET
     RESERVE      2 POINTERS
     10 BANNED      PIC 9(     1)
                    USAGE IS COMP
     10 BANNEE      PIC 9(     1)
                    USAGE IS COMP
                    RESULT
                    SUM
                    ON THIS RECORD USING
                    BANNED
                    BANNEF
                    CHECK
                    VALUE
```

```
                    NOT
                        1
10 BANNEF           PIC 9(     1)
                    USAGE IS COMP
                    RESULT
                    COUNT
                    SET SHIBNN
                    MEMBER BNNDSH
                    CHECK
                    VALUE
                    MAX
                        1
10 BANNEG           PIC 9(     1)
                    USAGE IS COMP
                    RESULT
                    PROCEDURE INTRSC
                    ON THIS RECORD USING
                    BANNED
                    OILTAN
10 CONVOY           PIC 9(     1)
                    USAGE IS COMP
10 HLLNMB           PIC 9(     6)
                    USAGE IS COMP
10 MERCHA           PIC 9(     1)
                    USAGE IS COMP
                    RESULT
                    PROCEDURE LNKE
                    ON THIS RECORD USING
                    TYPE
10 NMFHMP           PIC X(    24)
10 OILTAN           PIC 9(     1)
                    USAGE IS COMP
                    RESULT
                    PROCEDURE LNKB
                    SET CRGTYR
                    MEMBER CRGTYP
                    USING    CRGTYQ
10 OILTAO           PIC 9(     1)
                    USAGE IS COMP
                    RESULT
                    SUM
                    ON THIS RECORD USING
                    OILTAN
                    OILTAP
                    CHECK
                    VALUE
                    NOT
                        1
10 OILTAP           PIC 9(     1)
                    USAGE IS COMP
                    RESULT
                    COUNT
                    SET SHILTN
                    MEMBER LTNKRS
                    CHECK
                    VALUE
                    MAX
                        1
10 OILTAR           PIC 9(     1)
```

```
                    USAGE IS COMP
                    RESULT
                    PROCEDURE QEQUAL
                    SET SHILTN
                    MEMBER LTNKRS
                    USING    OILTAQ
     10 RURITA      PIC 9(     1)
                    USAGE IS COMP
     10 SAFESH      PIC 9(     1)
                    USAGE IS COMP
                    RESULT
                    PROCEDURE NOTPRC
                    ON THIS RECORD USING
                    BANNED
     10 SHIPNM      PIC X(    24)
     10 SHIPST      PIC 9(     1)
                    USAGE IS COMP
                    RESULT
                    SUM
                    ON THIS RECORD USING
                    BANNED
                    OILTAR
     10 TYPE        PIC X(    18)
     10 TYPESO      PIC 9(     1)
                    USAGE IS COMP
 RECORD NAME IS SHPTYP
*TYPES OF SHIPS
     LOCATION VIA QSYSTK   SET
     10 NMBRFS      PIC 9(     9)
                    USAGE IS COMP
                    RESULT
                    COUNT
                    SET SHPTYQ
                    MEMBER SHIPS
     10 TYPF        PIC X(    18)
                    RESULT
                    PROCEDURE QEQUAL
                    SET SHPSHI
                    MEMBER SHIPS
                    USING    TYPE
                    CHECK
                    VALUE
                    NOT
                    NULL
 RECORD NAME IS TYPSFH
     LOCATION VIA QSYSTL   SET
     10 TYPESP      PIC 9(     1)
                    USAGE IS COMP
                    RESULT
                    PROCEDURE ENUMPR
                    SET TYPSHI
                    MEMBER SHIPS
                    USING    TYPESO
                    CHECK
                    VALUE
                    NOT
                    0
     10 TYPSFI      PIC 9(     1)
                    USAGE IS COMP
```

```
SET SECTION
SET NAME IS CNTRYF
    MODE IS CHAIN
    ORDER LAST
    OWNER CONTRS
    MEMBER OFFCRS
      AUTOMATIC
     SELECTION
      CURRENT OF SET
SET NAME IS SHPSRG
    MODE IS CHAIN
    ORDER LAST
    OWNER CONTRS
    MEMBER SHIPS
      AUTOMATIC
     SELECTION
      CURRENT OF SET
SET NAME IS CONLNK
    MODE IS CHAIN
    ORDER SORTED
    OWNER CONVYS
    MEMBER LNKA
      AUTOMATIC
     DESCENDING
    CONVYT
    DUPLICATES FIRST
     SELECTION
      CURRENT OF SET
SET NAME IS CRGCRG
    MODE IS CHAIN
    ORDER LAST
    OWNER CRGTYS
    MEMBER CRGTYP
      AUTOMATIC
     SELECTION
      CURRENT OF SET
SET NAME IS NGNLNK
    MODE IS CHAIN
    ORDER LAST
    OWNER ENGINS
    MEMBER LINKA
      AUTOMATIC
     SELECTION
      CURRENT OF SET
SET NAME IS LSPLLS
    MODE IS CHAIN
    ORDER SORTED
    OWNER LTNKRS
    MEMBER NCDNTS
      AUTOMATIC
     DESCENDING
    LSPLLU
    DUPLICATES FIRST
     SELECTION
      CURRENT OF SET
SET NAME IS NSPCTO
    MODE IS CHAIN
    ORDER SORTED
    OWNER LTNKRS
```

```
        MEMBER NSPCTN
          AUTOMATIC
         DESCENDING
        NSPCTP
        NSPCTQ
        DUPLICATES FIRST
         SELECTION
          CURRENT OF SET
    SET NAME IS LSTNSP
        MODE IS CHAIN
        ORDER LAST
        OWNER MSTRCN
        MEMBER LTNKRS
          MANUAL
         SELECTION
          CURRENT OF SET
    SET NAME IS NCDLSP
        MODE IS CHAIN
        ORDER LAST
        OWNER NCDNTS
        MEMBER LSPLLT
          AUTOMATIC
         SELECTION
          CURRENT OF SET
    SET NAME IS NSPMST
        MODE IS CHAIN
        ORDER LAST
        OWNER NSPCTN
        MEMBER MSTRCN
          AUTOMATIC
         SELECTION
          CURRENT OF SET
    SET NAME IS CNTCTS
        MODE IS CHAIN
        ORDER LAST
        OWNER OFFCRS
        MEMBER LINKC
          AUTOMATIC
         SELECTION
          CURRENT OF SET
    SET NAME IS FFCLNK
        MODE IS CHAIN
        ORDER LAST
        OWNER OFFCRS
        MEMBER LINKC
          MANUAL
         SELECTION
          CURRENT OF SET
    SET NAME IS NVLVDC
        MODE IS CHAIN
        ORDER LAST
        OWNER OFFCRS
        MEMBER NCDNTS
          AUTOMATIC
         SELECTION
          CURRENT OF SET
    SET NAME IS OFFICR
        MODE IS CHAIN
        ORDER LAST
```

```
        OWNER OFFCRS
        MEMBER SHIPS
          AUTOMATIC
         SELECTION
          CURRENT OF SET
    SET NAME IS LNKHFC
        MODE IS CHAIN
        ORDER SORTED
        OWNER QENTRY
        MEMBER OFFCRS
          AUTOMATIC
         DESCENDING
        DTCMMS
        DUPLICATES FIRST
         SELECTION
          CURRENT OF SET
    SET NAME IS LSPLLV
        MODE IS CHAIN
        ORDER LAST
        OWNER QENTRY
        MEMBER LSPLLT
          AUTOMATIC
         SELECTION
          CURRENT OF SET
    SET NAME IS LTNKRU
        MODE IS CHAIN
        ORDER LAST
        OWNER QENTRY
        MEMBER LTNKRS
          AUTOMATIC
         SELECTION
          CURRENT OF SET
    SET NAME IS NSPCTR
        MODE IS CHAIN
        ORDER LAST
        OWNER QENTRY
        MEMBER NSPCTN
          AUTOMATIC
         SELECTION
          CURRENT OF SET
    SET NAME IS QSYSTD
        MODE IS CHAIN
        ORDER LAST
        OWNER QENTRY
        MEMBER CONTRS
          AUTOMATIC
         SELECTION
          CURRENT OF SET
    SET NAME IS QSYSTE
        MODE IS CHAIN
        ORDER LAST
        OWNER QENTRY
        MEMBER CONVYS
          AUTOMATIC
         SELECTION
          CURRENT OF SET
    SET NAME IS QSYSTF
        MODE IS CHAIN
        ORDER LAST
```

```
    OWNER QENTRY
    MEMBER CRGTYS
      AUTOMATIC
     SELECTION
      CURRENT OF SET
SET NAME IS QSYSTG
    MODE IS CHAIN
    ORDER LAST
    OWNER QENTRY
    MEMBER ENGINS
      AUTOMATIC
     SELECTION
      CURRENT OF SET
SET NAME IS QSYSTK
    MODE IS CHAIN
    ORDER LAST
    OWNER QENTRY
    MEMBER SHPTYP
      AUTOMATIC
     SELECTION
      CURRENT OF SET
SET NAME IS QSYSTL
    MODE IS CHAIN
    ORDER LAST
    OWNER QENTRY
    MEMBER TYPSFH
      AUTOMATIC
     SELECTION
      CURRENT OF SET
SET NAME IS CRGTYR
    MODE IS CHAIN
    ORDER LAST
    OWNER SHIPS
    MEMBER CRGTYP
      AUTOMATIC
     SELECTION
      CURRENT OF SET
SET NAME IS NCDNTT
    MODE IS CHAIN
    ORDER LAST
    OWNER SHIPS
    MEMBER NCDNTS
      AUTOMATIC
     SELECTION
      CURRENT OF SET
SET NAME IS SHIBNN
    MODE IS CHAIN
    ORDER LAST
    OWNER SHIPS
    MEMBER BNNDSH
      AUTOMATIC
     SELECTION
      CURRENT OF SET
SET NAME IS SHILNK
    MODE IS CHAIN
    ORDER LAST
    OWNER SHIPS
    MEMBER LNKA
      AUTOMATIC
```

```
      SELECTION
        CURRENT OF SET
SET NAME IS SHILTN
    MODE IS CHAIN
    ORDER LAST
    OWNER SHIPS
    MEMBER LTNKRS
      AUTOMATIC
     SELECTION
      CURRENT OF SET
SET NAME IS SHPNGN
    MODE IS CHAIN
    ORDER LAST
    OWNER SHIPS
    MEMBER LINKA
      AUTOMATIC
     SELECTION
      CURRENT OF SET
SET NAME IS SHPSHI
    MODE IS CHAIN
    ORDER LAST
    OWNER SHPTYP
    MEMBER SHIPS
      AUTOMATIC
     SELECTION
      CURRENT OF SET
SET NAME IS SHPTYQ
    MODE IS CHAIN
    ORDER LAST
    OWNER SHPTYP
    MEMBER SHIPS
      MANUAL
     SELECTION
      CURRENT OF SET
SET NAME IS TYPSHI
    MODE IS CHAIN
    ORDER LAST
    OWNER TYPSFH
    MEMBER SHIPS
      MANUAL
     SELECTION
      CURRENT OF SET
```

19. BIBLIOGRAPHY

ANSI/X3/SPARC (standards Planning and Requirements Committee), "Interim Report from the Study Group on Database Management Systems", FDT (Bulletin of ACM SIGMOD), Vol. 7, No. 2, 1975.

Abrial, J.R. "Data Semantics", in Database Management (eds. Klimbie,J. and Koffeman, K.) , North Holland, 1974.

Adiba, M., Delobel,C. and Leonard,M. "A Unified Approach for Modelling Data in Logical Database Design", Proc. IFIP-TC-2-WC, Black Forest, 1976.

Aho, A.V., Beeri, C. and Ullman, J.D. "The Theory of Joins in Relational Databases", ACM Trans. on Database Systems Vol. 4, 1979, pp. 297-314.

Aho, A.V., Sagiv, Y. and Ullman, J.D. "Efficient Optimization of a Class of Relational Expressions", ACM Trans. on Database Systems VOl. 4, 1979, pp. 435-454.

Aho, A.V., Sagiv, Y. and Ullman, J.D. "Equivalences Among Relational Expressions", SIAM Jnl. Comput. Vol. 8, 1979, pp. 218-246.

Ambrozy, D. "On Man-Computer Dialogue", Int. Jnl. Man-Machine Studies, Vol. 3, NO. 4, pp.375-383, 1971.

Anderson,J. "@@HELP: Online Documentation System", in Technical Papers, USE Spring Conf. 1979, Bladensburg, MD: USE, Inc., pp. 215-235.

Arlow, B.E. "Conceptual Data Modelling and S2K Database Design", The ASTUTE Conf. Proc. for Fall 1980, pp. 64-96.

Armstrong, W.W. "Dependency Structures of Data Base Relationships",
Proc. IFIP Congress 1974, North-Holland, Amsterdam, pp.580-583.

Armstrong, W.W. and Delobel,C. "Decompositions and Functional Dependencies in Relations", ACM Trans. on Database Systems Vol. 5, 1980, pp. 404-430.

Arnbirgm S., Elvers, E. and Svensson, P. "Design Specification for Datalab: A System for Data Analysis Based on Relational Model of Data", Res. Institute of National Defence, Sweden, N80-23041-0, November 1979.

Ash,W., Bobrow,R. Grignetti,M. and Hatley,A "intelligent On-Line Assistant and Tutor System, Technical Report No. 3607, Bolt Bernack and Newman, Inc., August, 1978.

Asprey, P.L. "Toward Automating the Database Design Process", Sandia Laboratories, SAnD-79-0858C, April 1979.

Bachman, C.W. "Data Structure Diagrams", Data Base Vol. 1, No. 2, 1969, pp. 4-10.

Bachman, C.W. "The Role Concept in Data Models", Proc. of Int. Conf. on Very Large Databases, Tokyo, Japan, October 1977.

Bachman, C.W. and Data,M. "The Role Concept in Data Models", Proc. 3rd Int. Conf. of Very Large Databases, 1977, pp.464-476.

Backus, J. "Can programming be liberated from the von Neumann style? A functional style and its algorithm of programs", Comm. of the ACM Vol. 21, No. 8, August, 1978, pp. 613-641.

Baldissera, C. et al "Interactive Specification and Formal Verification of User's Views in Database Design", Proc. 5th Int. Conf. on Very Large Databases, 1979, pp. 262-272.

Beeri, C. "On the Membership Problem for Functional and Multi-valued Dependencies in Relational Databases", ACM Trans. on Database Systems Vol. 5, 1980, pp.241-259.

Beeri, C. "On the Role of Data Dependencies in the Construction of Relational Database Schemas", Tech. Report 43, Department of Computer Science, The Hebrew University of Jerusalem, Jerusalem, Israel, 1979.

Beeri, C. et al "Equivalence of Relational Database Schemes", SIAM Jnl. of Computing, Vol. 10, pp. 352-370.

Beeri, C., Bernstein,P.A. and Goodman,N. "A Sophisticate's Introduction to Database Normalization Theory", Proc. 4th Int. Conf. on Very Large Databases, 1978, pp.113-124.

Beeri, C., Fagin, R. and Howard, J.H. "A Complete Axiomatization for Functional and Multivalued Dependencies in Database Relations", Proc. ACM SIGMOD, 1977, pp.47-61.

Beeri,C. and Bernstein,P.A. "Computational Problems Related to the Design of Normal Form Relational Schemas", ACM Trans. on Database Systems Vol. 4, 1979, pp.30-59.

Beeri,C. et al "On the Structure of Armstrong Relations for Functional Dependencies.", Technical Report RJ2901, IBM Res. Laboratory, San Jose CA, 1980.

Bell, A. and Quillian, M.R. "Capturing Concepts in a Semantic Net", Proc. Symposium on Associative Information Techniques, September 1968, Warren, Michigan, pp. 3-25.

Benci, E. et al "Concepts for the Design of a Conceptual Schema", in Modelling in Data Base Management Systems, (ed. Nijssen, G.M.), North-Holland, Amsterdam, 1976, pp.181-200.

Bennet, J.L. "User Interface in Interactive Systems, Annual Review of Information Sciences and Technology, (Ed. C. Cuadra), Vol. 7, Encyclopeadia Britannica, Chicago, Ill., pp 159 - 196, 1972.

Berild,S. and Nacmens, S. "CS4 - A Tool for Database Design by Infological Simulation", Proc. 3rd Int. Conf. on Very Large Databases, 1977, pp.533-544.

Bernstein, P.A. "Normalization and Functional Dependencies in the

Relational Database Model", Ph.D. Thesis, Department of Computer Science, University of Toronto, Toronto, 1975.

Bernstein, P.A. "Synthesizing Third Normal Form Relations from Functional Dependencies", ACM Trans. on Database Systems Vol. 1, 1976, pp. 277-298.

Bernstein, P.A. and Beeri,C. "An Algorithmic Approach to Normalization of Relational Database Schema", Technical Report CSRG-73, Comput. Systems Res. Group, University of Toronto, Toronto, 1976.

Bernstein, P.A. and Chiu,D.W. "Using semi-joins to Solve Relational Queries", Jnl. ACM Vol. 28, 1981, pp. 25-40.

Bernstein, P.A. and Goodman, N. "The Theory of Semi-Joins", Technical Report CCA-27-79, Computer Corporation of America, Cambridge MA, 1979.

Bernstein, P.A. and Goodman, N. "What does Boyce-Codd Normal Form Do?", Proc. 6th Int. Conf. on Very Large Databases, 1980, pp.245-259.

Bernstein, P.A., Blaustein,B.T. and Clarke, E.M. "Fast maintenance of Semantic Integrity Assertions Using Redundant Aggregate Data", Proc. 6th Int. Conf. on Very Large Databases, 1980, pp.126-136.

Bernstein, P.A., Swenson, J.R. and Tsichritzis,D.C. "A Unified Approach to Functional Dependencies and Relations", Proc. ACM SIGMOD Conf. May 1975, ACM New York NY, pp. 237-245.

Biller, H. and Neuhold, E.J. "Concepts for the Conceptual Schema", in Architecture and Models in Data Base Mangament Systems, (ed. Nijssen, G.M.), North-Holland, Amsterdam, 1977.

Biller, H. and Neuhold, E.J. "Semantics of Databases: The Semantics of Data Models", Information Systems, Vol. 3, No. 1, pp. 11 - 30, 1978.

Bjorner, D. et al "The Gamma Zero N-ary Relational Data Base Interface: Specification of Objects and Operations.", Technical Report RJ1200, IBM Res. Laboratory, San Jose CA, 1973.

Bodart, F. "Dynamics Specification of an Information System", IFIP Conf. on Formal Models and Practical Tools for Information System Design, Oxford, 1979.

Boehm, B. W., Seven, M. J. and Watson, R. A. "Interactive Problem-Solving - An Experimental Study of 'Lockout' Effects", AFIPS Conf. Proc., Vol. 36, 1971 SJCC, pp. 205 - 210.

Boies, S.J. "User Behaviour in an Interactive Computer System,: IBM Res. Report, PC4169, 1972.

Borkin, S.A. "Data Model Equivalence", Proc. 4th Int. Conf. on Very Large Databases, 1978, pp. 526-534.

Borkin, S.A. "Data Models: A Semantic Approach for Database Systems", MIT Press, Cambridge MA, 1980.

Bourne, T.J. "The Data Dictionary System in Analysis and Design", ICL Technical Jnl., Vol. 1, No. 3, pp. 292-298.

Boyce, R. F. and Chamberlin, D.D. "Using a Structured English Query Language as a Data Definition Facility", Technical Report RJ1318, IBM Res. Laboratory, San Jose CA, 1973.

Bracchi, G., Furtado, A.L. and Pelagatti, G. "Constraint Specification in Evolutionary Data Base Design", in Formal Models and Practical Tools for Information System Design, (ed. Schneider, H.-J.), North-Holland, Amsterdam, pp. 149-165.

Bracchi, R.F. Paolini,P. and Pelagatti, F. "Binary Logical Associations in Data Modelling", in Modelling in Data Base Management Systems"(ed. Nijssen, G, M.), North-Holland, Amsterdam, 1976, pp.125-148.

Bracchi, R.F., Paolini,P. and Pelagatti, F. "Mapping External Views to a Common Data Model", Inf. Syst. Vol. 3, 1978, pp. 141-151.

Brachman, R.J. "A Structural Paradigm for Representing Knowledge", Ph.D. Dissertation, Harvard University, 1977.

Brachman, R.J. "On the Epistemological Status of Semantic Networks", in Associatve Networks (ed. Findler, N.), Academic Press, New York NY, 1979, pp. 3 - 50.

Brachman, R.J. and Smith, B.C. "Special Issue on Knowledge Representation", SIGART Newsletter NO. 70, 1980.

Bradley, J. "File and Data Base Techniques", Holt, Rinehart and Winston Publishers, New York NY, 1982.

Brodie, M.L. " On the Conceptual Modelling of Behavior", University of Maryland, Department of Computer Science Technical Report, 1980.

Brodie, M.L. "Computer Science and Technology: Data Abstraction, Databases and Conceptual Modelling: an Annotated Bibliography", Maryland University, PB80-183833, May 1980.

Brodie, M.L. "On Modelling Behavorial Semantics of Data", submitted to the Seventh International Conf. on Very Large Databases, Cannes, France, 1981.

Brodie, M.L. "Specification and Verification of Data Base Semantic Integrity.", Ph. D. thesis, Department of Computer Science, University of Toronto, Toronto, 1978.

Brodie, M.L. "The Application of Data Types to Database Semantic Integrity", Inf. Syst. Vol 5, 1980, pp.287-296.

Brodie, M.L. and Zilles, S.N. (eds.), "Proc. of the Workshop on Data Abstraction, Databases and Conceptual Modelling", ACM SIGMOD Record Vol. 11, No. 2, ACM Order No. 474800, 1981.

Brown, A.P.G. "Modelling a Real World System and Designing a Schema to Represent It", in Proc. IFIP TC-2 Special Working Conf.,

"An In-Depth Technical Evaluation of Codasyl DDL", pp. 339-347.

Brown, R.M., "An Experimental Study of an On-line Man-Computer System." , IEEE Trans. on Electronic Computers, EC-14,1965, pp. 82-85.

Bubenko, J. A. Jr., "Validity and Verification Aspects of Information Modeling", Proc. 3rd Int. Conf. on Very Large Databases, 1977, pp. 556-565.

Bubenko, J.A. Jr. "Information Modeling in the Context of System Development", Proc. IFIP Congress 1980, pp. 395-411.

Bubenko, J.A. Jr. et al "From Information Structures to DBTG Data Structures", in Proc. Conf. on Data: Abstraction, Definition and Structure, ACM SIGPLAN/SIGMOD 1976, ACM New York NY, pp. 73-84.

Bubenko, J.A. Jr., Langefors, B. and Solvberg, A. (eds.), "Computer Aided Information Systems Analysis and Design", Studentlitteratur, Lund, Sweden, 1971.

Bubenko, J.A.Jr., "IAM: An Inferential Abstract Modeling Approach to Design of Conceptual Schemas", Proc. ACM SIGMOD, 1977, pp.62-74.

Buchman, A.P. and Dale, A.G. "Evaluation Criteria for Logical Database Design Methodologies", Computer-Aided Design, Vol. 11, No. 3, May 1979, pp. 121-126.

Buneman, P. and Frankel, R."FQL: A Functional Query Language", Proc. of ACM SIGMOD Int. Conf. on the Management of Data, Boston MA, 1979.

Cadiou, J.-M. "On Semantic Issues in the Relational Model of Data", in Proc. Int. Symp. Mathematical Foundations of Computer Science, Gdansk, Poland, Lecture Notes in Compter Sciences, Springer-Verlag, Heidelberg, September 1975.

Carbonell, J. R. "On Man-Computer Interaction:A Model and Some Related Issues.", IEEE Trans. on Systems Science and Cybernetics, SSC-5, 1969.

Carbonell, J.R. "Artificial InTelligence and Large Interactive Man Computer Systems", Proc. Systems, Man, and Cybernetics Symposium, 1971, pp. 167 - 173.

Card, S.K. "Studies in the Psychology of Computer Text Editing", Xerox Palo ALto Res. Centre, SSL-78-1, San Jose CA, August 1978.

Carlson, s. and Kaplan, R. "A Generalised Access Path Model and its Application to a Relational Database System", Proc. of ACM SIGMOD Int. Conf. on the Management of Data, Washington D.C., June 1976.

Cattell, R.G.G. "An Entity-Based Database User Interface", Proc. of ACM SIGMOD Int. Conf. on the Management of Data, Los Angeles CA, May 1980.

Chamberlin, D.D., Gray, J.N. and Traiger,I.L. "Views,

Authorization and Locking in a Relational Database System", Proc. AFIPS NCC Vol. 44, 1975, pp. 425-430.

Chamberlin,D. et al "SEQUEL 2: A Unified Approach to Data Definition, Manipulation, and Control", IBM Jnl. of Res. and Dev., Volume 20, Number 6, November 1976, pp. 560 - 575.

Chang, C.L. "A Hyper-Relational Model of Databases", IBM Res. Report RJ1634, San Jose CA, August 1975.

Chang, S.K. and Cheng, W.H. "Database SKeleton and its Applications to Logical Database Synthesis", IEEE Trans. on Software Engineering, Vol. SE-4, January 1978, pp. 18-30.

Chang, S.K. and Ke, J.S. "Database Skeleton and its Application to Fuzzy Query Translation", IEEE Trans. on Software Engineering, SE-4, January 1978, pp. 31-43.

Chen P.P. (ed.) "Entity-Relationship Approach to Sytems Analysis and Design", North-Holland, Amsterdam, 1980.

Chen, P. "The Entity-Relationship Model: Toward a Unified View of Data", ACM Transactions on Database Systems, Vol. 1, No. 1, March, 1976, pp. 9 - 36.

Chen, P.P. "The Entity-Relationship Model: A Basis for Enterprise View of Data", Proc. AFIPS NCC Vol. 46, 1977, pp. 77-84.

Chen, P.P. and Yao, S.B. "Design and Performance Tools for Data Base Systems", Proc. 3rd Int. Conf. on Very Large Databases, pp. 3-15.

Chen, P.P.S. "The Entity-Relationship Approach to Logical Database Design", Monograph No. 6, QED Information Sciences, Wellesley MA, 1978.

Childs, D.L. "Extended Set Theory - A general Model for very Large, Distributed, Backend Information Systems", Proc. 3rd Int. Conf. on Very Large Databases, 1977, pp. 28-46.

Codasyl Committee on Data System Languages, Codasyl Database Task Group Report, ACM New York NY, 1971.

Codasyl Data Description Language Journal of Development, Materal Data Management Branch, Department of Supply and Services, Ottawa, 1978.

Codasyl FORTRAN Data Base Facility Journal oF Development. Material Data Management Branch, Department of Supply and Services, Ottawa, 1977.

Codd, E.F. "A Relational Model of Large Shared Data Banks", Comm. of the ACM, Vol. 13, No. 6, June 1970.

Codd, E.F. "Extending the Database Relational Model", ACM Trans. on Database Systems, Vol. 4, No. 4, December 1979, pp. 397 - 434.

Codd, E.F. "Further Normalization of the Data Base Relational Model", in Data Base Systems, Courant Computer Science Symposium

6th (ed. Rustin, R.), Prentice-Hall, Englewood Cliffs NJ, 1972.

Codd, E.F. "Normalized Data Base Structure: A Brief Tutorial", Proc. ACM SIGFIDET Workshop on Data Description, Access and Control, 1971, pp. 1-17.

Codd, E.F. "Recent Investigations in Relational Database Systems", in Proc. 1974 IFIP Congress, North-Holland, Amsterdam, pp. 1017-1021.

Codd, E.F. "Seven Steps to Rendezvous with the Casual User", Database Management (eds. Klimbie, J. and Koffeman, K.), North Holland, 1974.

Codd, E.F. et al "RENDEZVOUS Version 1: An Experimental English-Language Query Formulation System for Casual Users of Relational Databases", IBM Res. Report RJ2144, San Jose CA, January 1978.

Cooley, R.E. "An Activation Model for Information Systems Specification", Syst. Arch., pp. 117-123.

Curtice, R.M. and Jones, P.E. Jr., "Key Steps in the Logical Design of Databases", Proc., NYU Symposium on Database Design, May 1976, pp. 51-66.

Date , C.J. "An Introduction to Database Systems", Addison-Wesley, 1977.

Davis, R. M. "Man-Machine Communication." in Annual Review of Information Science and Technology, Vol. 1, (ed. Cuadra, C.), Interscience Publishers, New York NY, 1966.

De Grenek, K.B.(ed.) "Systems Psychology", New York, McGraw Hill, 1970, chapter 10.

Deen, S.M. "Fundamentals of Database Systems", MacMillan Press, London, 1977.

Delobel, C. "A Theory About Data in an Information System", IBM Res. Report RJ 964, San Jose CA, January 1972.

Delobel, C. and Leonard, M. "The Decompostion Process in a Relational Model", Technical Report, Laboratoire d'Informatique, University of Grenoble, France, September 1974.

Duffy, G.F. and Gartner, F.P. "An On-Line Information System for MAnagement", Levin,M. and Schneider, L.S. "The Separation of Enterprise and Data Models", Database Res. Project, Sterling Systems Incorp., Golden Colorado.

Dyba, J.E. "Principles of Element Identification", Auerbach Data Base Management Services, Portfolio #23-01-01, 02, 1977.

Earnest, C.P. "A Comparison of the Network and Relational Data Structure Models", Technical Report Sciences Corp., El Segundo CA, 1974.

Earnest, C.P. "Selection and Higher Level Structures in Database

Systems", in Database Description, (Eds. Nissen, G.M. and Douque, B.C.M.), North-Holland, Amsterdam, 1975, pp. 215-237.

Eason, K.D. "Man-Computer Communication in Public and Private Computing", INFOTECH State-of-the-Art Report on Man/Computer Communication, Vol. 2, 1979.

El-Masri, R. and Wiederhold, G. "Properties of Relationships and their Representations", Computer Science Department, Stanford University, Stanford CA, 1979.

Engel, S. E. and Granda, R. E. "Guidelines for Man-Display Interfaces", IBM Poughkeepsie Laboratory Technical Report TR 00.2720, December 1975.

Eswaran, K.P. and Chamberlin, D.D. "Functional Specifications of a Subsystem for Database Integrity", Proc. of Int. Conf. on Very Large Databases, Framingham MA, September, 1975.

Everest, G.C. "Data Base Managment: Objectives, System Functions, and Administration", McGraw Hill, New York NY, 1977.

Fadous, R.Y. and Forsyth, J. "Finding Candidate Keys for Relational Data Bases", Proc. ACM SIGMOD Conf., May 1975, ACM New York NY, pp. 203-210.

Fagin, R. "A Normal Form for Relational Databases that is Based on Domains and Keys.", Tech. Report Rj2520, IBM Res. Laboratory, San Jose CA, 1979.

Fagin, R. "Horn Clauses and Data Dependencies", Proc. 12th ACM Symposium on Theory of Comput., 1980, pp. 123-134.

Fagin, R. "Multivalued Dependencies and a new Normal Form for Relational Databases", ACM Trans. on Database Systems, Vol. 2, No. 3, 1977, pp. 262-278.

Fagin, R. "Normal Forms and Relational Database Operators" in Proc. ACM SIGMOD 1979, pp. 152-160.

Fagin, R. "The Decompostion versus the Synthetic Approach to Database Design", Proc. 3rd Int. Conf. on Very Large Databases, October 1977, pp. 61-66.

Fahlman, S. "NETL: A System for Representing and Using Real-World Knowledge", MIT Press, 1979.

Fahlman, S. E. "A System for Representing and Using Real-World Knowledge", Ph.D. Thesis, Artificial Intelligence Lab., MIT., 1977.

Falkenberg, E.D. "Conceptualizatin of Data", Infotech State-of-the-Art Report, 1980.

Frankel, A.A., Bar-Hiller, Y. and Levy, A. "Foundations of Set Theory", North-Holland, AMsterdam, 1973.

Frankel, R.E. "FQL - The Design and Implementation of a Functional Database Query Language", Wharton School, AD-A081786-6, May 1979.

Fried, B. "On the User's Point of View", in Interactive Systems for Experimental Applied Mathematics (eds. Klere,M. and Reinfelds,I.), New York: Plenum, 1971, pp. 91-107.

Fry, J.P., Smith, D.C.P. and Taylor, R.W. "An Approach to Stored Data Definition and Translation", Proc. ACM SIGFIDET Workshop on Data Description, Access and Control, 1972, pp. 13-55.

Gaines, B. R. and Facey, P. V. "Some Experience in Interactive System Development and Application", Proc. IEEE, Vol. 63, No. 6, June 1975, pp. 894-911.

Gaines, C.P. "Data Design in Structured Systems Analysis", Tutorial on Software Design Techniques, (eds. P. Freeman and Wasserman, A.I.), 1980.

Gambino, T.J. and Gerritsen, R. "A Database Design Decision Support System", Proc. 3rd Int. Conf. on Very Large Databases, pp. 534-545.

Gammill, R.C. and Shukiar, H.J. "An Interactive System for Aiding Management Decision Making", AFIPS 1977, pp. 753-759.

Gerritsen, R. "A Preliminary System for the Design of DBTG Data Structures", Comm. of the ACM Vol. 18, 1975, pp.551-557.

Gerritsen,R. "Tools for Automation of Database Design", Proc. NYU Symposium of Database Design, 1798, pp. 1997.

Goldstein, I. "Research Directions in Database Management", in Research Directions in Software Technology (P. Wegner ed.), MIT Press, 1979.

Gotlieb, L.R. "Computing Joins of Relations", in Proc. 1975 ACM SIGMOD Int. Conf. on Management of Data, ACM New York NY, pp. 55-63.

Gottlieb, C.C. and Tompa,F.W. "Choosing a Storage Schema", Acta Informatica Vol. 3, pp. 297-319.

Grabowski, H. and Eigner, M. "Semantic Datamodel Requirements and Realization with a Relational Datastructure", Computer-Aided Design, Vol. 13, No. 3, May 1979, pp. 158-167.

Griffith, R.L. "Three Principles of Representation for Semantic Networks", ACM Trans. on Database Systems, Vol. 7, No. 3, September 1982, pp. 417-442.

Grignetti, M.C., Hausmann, C. and Gould, L. "An Intelligent Online Assistant and Tutor - NLS Scholar", AFIPS 1975, pp. 775-781.

Grotenhuis, F. and Brock, J. "A Conceptual Model for Information Processing" in Proc. of the IFIP Working Conf. on Modeling in DBMS, Freudenstadt, 1976. Raver, N. and Hubbard, G.U. "Automated Logical Data Base Design", IBM Syst. Jnl., Vol. 16, No. 3, 1977.

Gunther, H, Krieger, R. and Lausen, G, "A Method for Interactive Conceptual Database Design", Systems Architecture, Westbury House, 1981, pp.177-187.

Guttag, J.V. and Horning, J.J. "Formal Specification as a Design Tool", Proc. 7th ACM Symposium on Principles of Programming Languages, pp. 251-261.

Hainaut,J.L. and Lecharlier,B. "An Extensible Semantic Model of Data Base and its Data Language", Proc. IFIP Congress 1974, North-Holland, Amsterdam, pp. 1026-1030.

Hall, D. J. "Man-Machine Projects at SRI", Int. Jnl. of Man-Machine Studies, Vol. 2, No. 4, 1970, pp. 363-394.

Hall,P., Owlett,J. and Todd,S.J.P. "Relations and Entities", in Modelling in Data Base Management Systems (Ed. Nijssen, G.M.), North-Holland, AMsterdam, 1976, pp. 201-220.

Hammer, M. "Data Abstractions for Data Bases", Proc. Conf. on Data: Abstraction, Definition and Structure, ACM FDT Vol. 8, No. 2, 1976, pp. 58-59.

Hammer, M. "Reseach Directions in Database Management", in Research Directions in Software Technology (P. Wegner ed.), MIT Press, 1979.

Hammer, M. "Self-Adaptive Automatic Database Design", AFIPS 1977, pg. 123.

Hammer, M. and Berkowitz, B. "DIAL: A Programming Language for Data-Intensive Applications", Proc. of ACM SIGMOD Int. Conf. on the Management of Data, Austin TX, May 1978.

Hammer, M. and McLeod, D. "A Framework for Database Semantic Integrity", Proc. of Second Int. Conf. on Software Engineering, San Francisco CA, October 1976.

Hammer, M. and McLeod, D. "Database Description with SDM: A Semantic Database Model", ACM Trans. on Database Systems, Vol. 6, No. 3, September 1981, pp. 351 - 386.

Hammer, M. and McLeod, D. "On the Architecture of Database Management Systems", Infotech State-Of-the-Art Report on Data Design,
Pergamon Infotech Ltd., 1980.

Hammer, M. and McLeod, D. "Semantic Integrity in a Relational Database System, Proc. of Int. Conf. on Very Large Databases, Framingham MA, September, 1975.

Hammer,M. and McLeod,D. "The Semantic Database Model: A Modelling Mechanism for Database Applications", Proc. of ACM SIGMOD Int. Conf. on the Management of Data, Austin TX, May 1978.

Hansen, W.J. "User Engineering Principles for Interactive Systems", AFIPS Conf. Proc., Vol. 39, 1971 FJCC, AFIPS Press, Montvale N.J., 1971, p. 523 - 532.

Hawryszkiewycz, I.T. "Alternate Implementations of the Conceptual Schema", Inf. Syst. Vol. 5, 1980, pp. 203-217.

Hawryszkiewycz, I.T. "Semantics of Data Base Systems", MIT Project MAC Report MAC TR-112, Cambridge MASS., December 1973.

Hayes, P.J. "On Semantic Nets, Frames and Associations", Proc. 5th Int. Joint Conf. on Artificial Intelligence, 1977, pp.99-107.

Heimbigner, D. and McLeod, D. "Federated Information Bases", Infotech State-of-the-Art Report, Pergamon Infotech Ltd., 1982.

Hendrix, G., Sacerdoti, E., Sagalowicz,D. and Stocum,J. "Developing a Natural Language Interface to Computer Data", Proc. of the Int. Conf. on Very Large Databases, Tokyo, Japan, October, 1977.

Hendrix, G.G. "Expanding the Utility of Semantic Networks through Partitioning", Proc. 4th Int. Joint Conf. on Artificial Intelligence, 1975, pp. 115-121.

Hendrix, G.G. "Some General Comments on Semantic Networks", Proc. 5th Int. Joint Conf. on Artificial Intelligence, 1977, pp. 984-985.

Hendrix, G.G. "The Representation of Semantic Knowledge", in Speech Understanding Research: Final Technical Report (ed. Walker, D.E.), Stanford Res. Inst., Menlo Park CA, 1976, Chapter 5.

Herot, C. "Spatial Management of Data", ACM Trans. on Database Systems, Vol. 5, No.4, December, 1980, pp. 493 - 514.

Heyderhoff,P. "Comment on Database Design: Towards an Integrated Methodology by Bo Sundgren", in Database Management (eds. Weber,H. and Wasserman, A.I.), North-Holland, Proc. 4th Conf. on Very Large Databases, 1978.

Honeyman, P. "Functional Dependencies and the Universal Instance Property in the Relational Model of Database Systems", Ph.D.Thesis, Department of Electrical Engineering and Computer Science, Princeton University, Princeton NJ, 1980.

Housel, B.C. et al, "DEFINE - A Nonprocedural Language for Defining Informati Easily", Proc. ACM Pacific 1975, pp. 62-70.

Housel, B.C., Waddle, V. and Yao, S.B. " The Functional Dependency Model for Logical Database Design", Proc. of the 5th Int. Conf. on Very Large Databases, Rio de Janeiro, October, 1979.

Hoyer, R. "User Participation - Why is Development so Slow?", in Information Systems Environment, (Eds. Lucas, H.C. et al), North-Holland Amsterdam, xix + 345 pp..

Hubbard, G.U. "Computer-Assisted Logical Database Design", Computer-Aided Design, Vol. 11, No. 3, May 1979, pp. 169-179.

Hubbard, G.U. "Techniques for Automated Logical File Design", NYU Sumposium of Database Design, May 1978.

IBM, Data Base Design Aid: Designer's Guide IBM #GH10-1627.

Jankowitz, H., Kilfoil, P.W. and Rabkin, I. "DBTRACE: A High-Level Database Performance Monitoring and Debugging System", Computer Science Department, University of Cape Town, Technical Report, 1982.

Jardine, D.A. and Steele, T.B. Jr. "ISO Report on Concepts for Conceptual Schemas", Proc. of the 6th Int. Conf. on Very Large Data Bases, Montreal, 1980, pp. 321-325.

Johnson, C. I. "Principles of Interactive Systems", IBM Syst. Jnl., Vol. 7, No.s 3 and 4, pp. 147-173, 1968.

Kahn, B.K. "A Method for Describing Information Required by the Database Design Process", Proc. ACM SIGMOD, 1976, pp. 53-64.

Kahn, B.K. "A Structured Logical Database Design Methodology", Proc. NYU Symposium on Database Design, 1978, pp. 15-24.

Kalinichenko, L.A. "Data Model Transformation Method Based on Axiomatic Model Extension", Proc. 4th Int. Conf. on Very Large Databases, 1978, pp.549-555.

Kameny, i. "EUFID: The End-User Friendly Interface to Data Mangement Systems", Proc. of Int. Conf. on Very Large Databases, West Berlin, West Germany, September 1978.

Katz, R.H. "Database Design and Translation for Multiple Data Models", Ph.D Thesis, Memo. No. UCB-ERL M80-24, Electronics Res. Laboratory, College of England, University of California, Berkeley CA, 1980.

Kennedy, T. C. S. and Edmunds, R. "An Examination of Training Problems with Naive Computer Users", in Proc. Eur. Computing Congress (EUROCoMP74), May 1974, pp. 411-425.

Kennedy, T.C.S. "Some behaviour Factors Affecting the Training of Naive Users of an Interactive Computer System", Int. Jnl. of Man-Macine Studies, Vol. 7, pp. 817 - 834, 1975.

Kennedy, T.S.C. "Design of Interacative Procedures for Man-Machine Communitcation", Int. Jnl. Man-Machine Studies, Vol. 6,pp 309 334, 1974.

Kent W. "Limitations of Record-Based Information Models", ACM Trans. on Database Systems, Vol. 4, No. 1, March 1979, pp. 107-131.

Kent, W. "A Primer of Normal Forms", IBM Technical Report TR 02.600, San Jose CA, December 1973.

Kent, W. "Describing Information (Not Data, Reality?)", Tech. Report 03.012, IBM General Products Division, Palo Alto CA, 1976.

Kent, W. "New Criteria for the Conceptual Model", in Systems for Systems for Large Databases, (eds. Lockemann and Neuhold), North-Holland, 1977, Proceedings of the 2nd Conf. on Very Large Databases, Belgium, 1976.

Kent, W. Data and Reality. North Holland, 1978.

Kershberg, L. Klug, A. and Tsichritzis,D.C. "A Taxonomy of Data Models", in Systems for Large Data Bases (eds. Lockeman,P.C. and Neuhold,E.J.), North-Holland, Amsterdam, 1976.

Kershberg, L. and Pacheco, J.E.S. "A Functional Data Base Model", Monograph Series in Computer Science and Computer Applications Dep. de Informatica, Pontificia University, Catolica do Rio De Janeiro, Brazil, 1975.

King, M.C.F., Naude, G. and Von Solms, S.H. "Database Design: Choice of a Methodology", Quaestiones Informaticae, Vol. 1, September 1979, pp. 16-22.

King, R. "The Event Database Specification Model", Ph.D. Thesis, Computer Science Department, University of Southern California, Los Angeles CA, 1981.

King, R. and McLeod, D. "An Approach to Database Design and Evolution", in Database Modelling, (eds. Brodie,M. , Mylooulos,J. and SChmidt,J.), 1982.

King, R. and McLeod, D. "Semantic Database Models", in Database Design (ed. Yao, S. B.), Prentice Hall, 1982.

King, R. and McLeod, D. "The Event Database Specification Model (A Preliminary Report)", in Database Modelling, (eds. Brodie,J., Mylopoulos,J. and Schmidt, J.) , 1982.

King, R. and McLeod, D. "The Event Database Support Model", In Conceptual Modelling (eds. Brodie,M., Mylopoulos,J. and Schmidt,J.), Springer Verlag, 1982.

Klug, A. and Tsichritzis, D.C. "Multiple View Support within the ANSI/SPARC Framework", Proc. 3rd Int. Conf. Very Large Databases, 1977, pp.477-488.

Kroenke, D. "Database Processing: Fundamentals, Modeling, Applications", Science Res. Associates, Palo Alto CA, 1977.

Langefors, B. "Control Structure and Formalized Information Analysis in an Organization", in Information Systems and Organizational Structure, (eds. Grochla, E. and Szyperski, N.), Walter de Gryter, New York NY, 1975, pp. 311-322.

Langefors, B. "Infological Models and Information User Views", Inf. Syst. Vol. 5, pp. 17-32.

Langefors, B. "Information Systems Theory", Inf. Syst. Vol. 2, 1977, pp. 207-219.

Langefors, B. "Information Systems", Proc. IFIP Congress 1974, North-Holland, Amsterdam, pp.937-945.

Langefors, B. "Management Information Systems Design", IAG Quart. Vol. 2, No. 4, 1969, pp. 7-17.

Langefors, B. "Some Approaches to the Theory of Information Systems", BIT Vol. 3, 1963, pp. 1-79.

Lawler, E.L. and Wood, D.E. "Branch and Bound Methods: a Survey", Op. Res., Vol. 14, No. 4, pp. 699-719.

Ledgard, H.F. and Taylor, R.W. "Two Views of Data Abstraction",

Comm. of the ACM Vol. 20, No. 6, pp. 382-384.

Lee, R.M. and Gerritsen, R. "Extended Semantics for Generalization Hierarchies", Proc. ACM SIGMOD Int. Conf. on the Management of Data, Austin TX, June 1978.

Levesque, H. "A Procedure Approach to Semantic Networks", M.Sc. Thesis, Department of Computer Science, University of Toronto, 1977.

Levesque, H.J. and Mylopoulos, J. "A Procedural Semantics for Semantic Networks", in Associative Networks (ed. Findler, N.), Academic Press, New York NY, 1979, pp. 93-120.

Lewis, E., Sekino,L.C. and Ting,P.D. "A Canonical Representation for the Relational Schema and Logical Data Independence", Proc. COMPSAC 1977, IEEE Computer Software and Applications Conf., 1977, pp. 276-280.

Licklider, J. C. R. "Man-Computer Communication", in Annual Review of Information Science and Technology, Vol. 3, (ed. Cuadra, C. A.), Encyclopeadia Britannica, Chicago Ill, 1968.

Licklider, J.C.R. "Man-Computer Symbiosis", IRe Trans. Hum. Factors Electron. Vol. HFE-1, pp. 4 - 11, March 1960.

Licklider, J.C.R. and Taylor, R. W. "The Computer as a Communication Device", Sci. Technology, April 1968, pp. 21-31.

Lien, Y.E. "Multivalued Dependencies with null values in relational data bases", Proc. 5th Int. Conf. on Very Large Databases, 1979, pp.61-74.

Lien, Y.E. "On the Equivalence of Database Models", Database Res. Report 3, Bell Labs, Holmdel NJ, 1980.

Lindencrona-Ohlin, E. "A Study on Conceptual Data Modeling", Ph.D. Thesis, Department of Computer Science, Chalmers University of Technology, Gothenburg, Sweden, 1979.

Lindgren, P. "Basic Operations on Information as a Basis for Database Design", Proc. IFIP Congress 1974, North-Holland, Amsterdam, 1974.

Ling, T.-W., Tompa, F.W. and Kameda,T. "An Improved Third Normal Form for Relational Databases", ACM Trans. on Database Systems, Vol. 6, No. 2, June 1981, pp. 329-346.

Liskov, B. et al "Abstraction Mechanisms in CLU", C.A.C.M., Vol. 12, No. 8, November 1977, pp. 564 - 576.

Lockeman, P.C. et al, "Data Abstractions for Database Systems", ACM Trans. on Database Systems Vol. 4, 1979, pp. 60-75.

Lucas, H.J. Jr. "The Analysis, Design and Implementation of Information Systems", McGraw-Hill, New York NY, 1981.

Luk,W.S. "On Data Dependency Structures of Relational Databases", Inf. Syst. Vol. 6, 1981, pp.23-29.

Lum, V.Y. et al "1978 New Orleans Data Base Design Workshop Report", Proc. 5th Int. Conf. Very Large Databases, 1979, pp. 328-350.

Lundeberg, M., Goldkuhl, G. and Nilsson, A. "Information Systems Development - A Systematic Approach", Prentice-Hall, Englewood Cliffs NJ, 1981.

Lyon, K. "Introduction to Data Base Design", Wiley Interscience, Div. of John Wiley and Sons, New York NY, 1971.

Mahood, C.E. "APL Relational Database Design Aid", APL 1980, pp. 229-235.

Maier, D., Mendelzon, A.O. and Sagiv, Y. "Testing Implications of Data Dependencies", ACM Trans. on Database Systems Vol. 4, 1979, pp.455-469.

March, S.T. and Severance D.G. "A Mathematical Modeling Approach to the Automatic Selection of Database Designs", Proc. ACM SIGMOD, 1978, pp. 52-65.

Martin, J. "Design of Man-Computer Dialogue", Prentice Hall, Englewood Cliffs, New Jersey, 1973.

Martin, J.T. "Computer Data-Base Organisation", Prentice-Hall, Englewood Cliffs NJ, 1975.

McCormick, E. J. "Human Factors Engineering", 3rd. ed., McGraw Hill, New York NY, 1970.

McLeod, D. "High-Level Definition of Abstract Domains in a Relational Database System", Jnl. of Computer Languages, Vol. 2, No. 3, 1977.

McLeod, D. "On Conceptual Database Modelling", Proc. of Workshop on Data Abstraction, Databases and Conceptual Modelling, Pingree Park Co, June 1980.

McLeod, D. and King, R. "Applying a Semantic Database Model", Proc. of Int. Conf. on the Entity-Relationship Approach to Systems Analysis and Design, Los Angeles CA, December 1979.

McLeod, D. and Smith, J.M. "Abstraction in Databases", Proc. of Workshop on Data Abstraction, Databases and Conceptual Modelling, Pingree Park Co, June 1980.

Mealy, G.H. "Another Look at Data", AFIPS Conf. Proc., Fall Joint Computer Conf., Vol. 31, 1967, pp. 525-534.

Mellberg, K. "Some Swedish Experience in the development of Interactive Computer Systems", Int. Jnl. Man-Machine Studies, Vol. 1, No. 1, pp. 51-72, 1969.

Melli, L. "Conceptual Data Model Methodology", Internal Report, Ontario Hydro, Toronto, 1980.

Mendelzon, A.O. "On Axiomatizing Multivalued Dependencies in Relational Databases", Jnl. ACM Vol. 26, 1979, pp. 37-44.

Mendelzon, A.O. and Maier, D. "Generalized Mutual Dependencies and the Decomposition of Database Relations", Proc. 5th Int. Conf. on Very Large Databases, 1979, pp. 75-82.

Metzer, H.S. "Current Concepts in Database Design", IBM Corp., San Jose CA.

Miller, G. A., "The Magic Number Seven, Plus or Minus Two: Some Limits on our Capacity for Processing Information", Psych. Rev., Vol. 63, No. 2, 1956, pp. 81-97.

Miller, L. A. and Thomas, J. C. "Behavioral Issues in the Use of Interactive Systems", Int. Jnl. Man-Machine Studies, Vol. 9, 1977, pp. 509 - 536.

Miller, L. H. "A Study in Man-Machine Interaction", AFIPS Conf. Proc. Vol. 46, 1977 NCC, pp. 409-421.

Miller, R. B. "Archetypes in Man-Computer Problem Solving", IEEE Trans. on Man-Machine Systems, Vol. MMS-10, December 1969, pp. 219 - 241.

Mills, R. G. "Man-Computer Interaction - Present and Future", IEEE Int. Convention Record, Vol. 14, Santa Monica CA, 1967.

Mills, R. G. "Man-Machine Communication and Problem Solving", in Annual Review of Information Science and Technology, Vol. 2, (ed. Cuadra, C.), Interscience Publishers, New York NY, 1967.

Mitoma, M.F. and Irani,K.B. "Automatic Database Schema Design and Optimization", in Proc. of the Int. Conf. on Very Large Databases, 1975, ACM New York NY , pp. 286-321.

Mylopoulos, J. "An Overview of Knowledge Representation", Proc. of Workshop on Data Abstraction, Databases, and Conceptual Modelling, Pingree Park Co, June 1980.

Mylopoulos, J. and Wong, H.K.T. "Some Features of the TAXIS Data Model", Proc. of 6th Int. Conf. on Very Large Databases, 1980, pp. 399-410.

Mylopoulos,J., Bernstein, P.A. and Wong, H.K.T. "A Language Facility for Designing Database-Intensive Applications", ACM Trans. on Database Systems, Vol. 5, No. 2, June 1980, pp. 185 207.

Mylopouplos.J. et al "TORUS: A Step Towards Bridging the Gap Between Databases and the Casual User", Information Systems, Vol. 2, No. 2, 1976, pp. 49 - 64.

Navathe and Schkolnick "View Representation in Logical Database Design", Proc. of ACM SIGMOD Int. Conf. on the Management of Data, Austin TX, June 1978.

Navathe, S.B. "ADMIRE: A Data Model to Incorporate Database Restructuring Effectively", Grad. School of Business, New York University, New York NY, 1977.

Navathe, S.B. "An Intuitive Approach to Normalize Network Structured Data", Proc. 6th Int. Conf. on Very Large Databases,

1980, pp. 350-358.

Neuhold, E.J. "The Use of Formal Description Techniques in Large Data Banks", Report, Institut fur Angewandte Informatik, Univ. of Karlsruhe, Germany, 1972.

Newell, A. and Simon, H. A. "Human Problem Solving.", Englewood Cliffs N.J., Prentice Hall, 1972.

Nicholas, J.M. "First Order Logic Formalization for Functional, Multi-valued and Mutual Dependencies", Proc. ACM SIGMOD, 1978, pp.40-51.

Nicholas, J.M. "Mutual Dependencies and Some Results on Undecomposable Relations", Proc. 4th Int. Conf. on Very Large Databases, 1978, pp. 360-367.

Nickerson, R. S. "Man-Computer Interaction: A Challenge for Human Factors Research", IEEE Trans. Man-Machine Studies, MMS-10, 1969, p. 164.

Nievergelt, J. and Weydert, J. "Sites, Models and Trails: Telling the User of an Interactive System Where he is, What he can do and How to get to Places", Proc. IFIP WG 52 Workshop on Methodology of Interaction, Seillac, May 1979.

Nijssen, G.M. "Current Issues in Conceptual Schema", in Architecture and Models in Data Base Mangement Systems (Ed. NIjssen, G.M.), North-Holland, Amsterdam, 1977, pp. 31-65.

Nijssen, G.M. "Data Structuring in the DDL and Relational Model", in Data Base Management, (eds. Klimbie, J.W. and Koffeman, K.L.), North-Holland, Amsterdam, 1974, pp. 363-384.

Olle, T.W. "Equivalence of Alternative Conceptual Schema Models", T. William Olle Associates Ltd., Surrey, England, 1979.

Olle, T.W. "MIS: Data Bases", Datamation Vol. 16, 15, Nov. 1970.

Olle, T.W. "Multistage Data Definition in a Multi-Component DBMS Architecture", in Databases: Improving Usability and Responsiveness, (ed. Shneiderman, B.), Academic Press, 1978.

Orrm W,D.D. (ed.), Conversational Computers, New York, Wiley, 1968.

Osborn, S.L. "Testing for Existence of a Covering Boyce-Codd Normal Form", computer SCience Department, University of Western Ontario, London, 1978.

Palme, J. "A Human-Computer Interface for Non-Computer Specialists", Software Practice and Experience, Vol. 9, 1979, pp. 741-747.

Palmer, I. "Record Subtype Facilities in Database Systems", Proc. of the 4th Int. Conf. on Very Large Databases, West Berlin, West Germany, September, 1978.

Paolini, P. "Abstract Data Types and Data Bases", Proc. of

Workshop on Data Abstraction, Databases and Conceptual Modelling, Pingree Park Co, June 1980.

Peace, D.M.S.and Easterby, R.S. "The Evaluation of User Interaction with Computer-Based Management Information Systems", Hum. Factors, Vol. 15, No. 2, 1973, pp. 163-177.

Pirotte, A. "The Entity-Property-Association Model: An InformationOriented Database Model", Technical Report, M.B.L.E. Res. Laboratory, Brussels, Belgium, 1977.

Press, L. "Toward Balanced Man-Machine Systems", Int. Jnl. Man-Machine Studies, Vol 3, NO. 1, pp. 61 - 73, 1971.

Pulfer, J.K. "Man-Machine Interaction in Creative Applications", Int. Jnl. Man-Machine Studies, Vol. 3, No. 1, pp. 1-11, 1971.

Relles, N. "The Design and Implementation of User-Oriented Systems", Computer Sciences Technical Report #357, University of Wisconsin-Madison, July 1979.

Rissanen, F. "Independent Components of Relations", ACM Trans. on Database Systems, Vol. 2, 1977, pp. 317-325.

Roberston,G. , Newell,A. and Ramakrishna,K "ZOG: A Man-Machine Communication philosophy", PIttsburgh PA, Carnegie-Mellon University, Department of Computer Science, August 1977.

Rolland, C. et al "Tools for Information System Dynamics Management", 5th Conf. on Very Large Databases, Rio de Janeiro 1979.

Ross, D.T. "Structured Analysis (SA): A Language for Communicating Ideas", IEEE Trans. on Software Eng., Vol. SE-3, No. 1, January 1977.

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

Rothnie, J.B. and Hardgrave, W.T. "Data Model Theory: A Beginning", Technical Report 10, Department of Inf. Syst. Management, University of Maryland, College Park MD, 1976.

Roussopoulos, N. "A Semantic Network Model for Data Bases.", Ph.D. Thesis, Department of Computer Science, University of Toronto, Toronto, 1976.

Roussopoulos, N. "ADD: Algebraic Data Definition", Proceedings of 6th Texas Conf. on Computing Systems, Austin TX, November, 1977.

Roussopoulos, N. "Algebraic Data Definition", in Proc. 6th Texas Conf. on Computing Systems, Austin TX, November 1977.

Roussopoulos, N. "Tools for designing Conceptual Schemata of Databases", Computer-Aided Design, Vol. 11, No. 3, May 1979, pp. 119-120.

Roussopoulos, N. and Mylopoulos, J. "Using Semantic Networks for Data Base Management", Proc. 1st Int. Conf. on Very Large

Databases, 1975, pp.144-172.

Rowe, L.A. and Shoen, K.A. "Data Abstraction, Views and Updates in RIGEL", Proc. of ACM SIGMOD Int. Conf. on the Management of Data, Boston MA, JUne 1979.

Sackman, H. "Man-Computer Problem Solving", Auerbach Publishers Inc., Princeton N.J., 1970.

Sackman, H. "Experimental Analysis of Man-Computer Problem-Solving", Human Factors, Vol. 12, 1970, pp. 187-201.

Sadri, F. and Ullman, J.D. "A Complete Axiomatization for a Large Class of Dependencies in Relational Databases" Proc. 12th ACM Symposium on Theory of Comput., pp. 117-122.

Sagiv, Y. et al "An Equivalence Between Relational Database Dependencies and a Fragment of Propositional Logic", Jnl. ACM 28, No. 2, April, 1981, pp. 435-453.

Sakai, H. "A Unified Approach to the Logical Design of a Hierarchical Data Model", in Entity-Relationship Approach to Systems Analysis and Design, (ed. Chen, P.P.), North-Holland, Amsterdam, 1980, pp. 61-74.

Schank, R. C. "Identification of Conceptualizations Underlying Natural Language", Computer Models of Thought and Language (eds. Schank, R. C. and Colby, K. M.), W. H. Freeman, 1973.

Schapiro, S.C. and Kwawsny,S.C. "Interactive Consulting via Natural Language", Comm. of the ACM Vol. 18, No. 15, May 1975, pp. 459-462.

Schkolnick, M. "Physical Database Design Techniques", Proc. NYU Symposium on Database Design, 1978, pp. 99-109.

Schmid, H.A. "An Analysis of some Constructs for Conceptual Models", in Architecture and Models in Data Base Management Systems (Ed. Nijssen, G.M.), North-Holland, Amsterdam, 1977, pp. 119-148.

Schmid, H.A. and Swenson, J.R. "On the Semantics of the Relational Data Model", Proc. of ACM SIGMOD Int. Conf. on the Management of Data, San Jose CA, May 1975.

Schmidt, J.W. "Type Concepts for Database Definition", in Databases: Improving Usability and Responsiveness (ed. Shneiderman, B.), Academic Press, New York NY, 1978, pp. 215-244.

Shneiderman, B. "Software Psychology: Human Factors in Computer and Information Systems", Winthrop Publishers, 1980.

Schubert, L.K. "Extending the Expressive Power of Semantic Networks", Artificial Intelligence Vol. 7, 1976, pp. 163-198.

Sciore, E. "Real-World MVD's", Proc. ACM SIGMOD, 1980, pp. 121-132.

Sciore, E. "The Universal Instance and Database Design", Tech.

Report 271, Department of Electrical Engineering and Computer Science, Princeton University, Princeton NJ, 1980.

Senko, M. E. "Conceptual Schemas, Abstract Data Structures, Enterprise Definitions", Proc. of ACM Int. Computing Symposium, Belgium, April 1977.

Senko, M. E. "Information Systems: Records, Relations, Sets, Entities and Things", Information Systems, Vol. 1, No. 1, PP. 3 - 14, 1975.

Senko, M.E. "Conceptual Schemas, Abstract Data Structures, Enterprise Descriptions", in Proc. ACM Int. Computing Symposium, Belgium, April 1977.

Senko, M.E. "Data Structures and Data Accessing in Database Systems Past, Present and Future", IBM Systems Jnl. Vol. 16, 1977, pp. 208-257.

Senko, M.E. et al "Data Structures and Accessing in Data-base Systems", IBM Systems Jnl. Vol. 12, 1973, pp. 30-93.

Shen, S.N.T. "A Semantic Approach in Designing Relational Databases", ACM 1978 Annual Conf. Proc., pp. 596-601.

Shen, S.N.T. and Krulee, G.K. "Solving Linear Programming Problems Stated in English by Computer", ACM Conf. Proc., 1973, pp. 299-303.

Shepard, J. Automation and Alienation, M.I.T. Press, 1971.

Sheppard, D.L. "Data Base Design Methodology - Parts I and II", Auerbach Data Base Management Services, Portfolio #23-01-01, 02, 1977.

Sheridan, T. B. and Ferrell, W. R. "Man-Machine Systems", Cambridge Mass., M. I. T. Press, 1974.

Shipman, D. "The Functional Data Model and the Data Language DAPLEX", ACM Trans. on Database Systems, Vol. 6, No. 1, March 1981, pp. 140 - 173.

Shneiderman, B. "Human Factors Experiments in Designing Interactive Systems", IEEE Computer, Vol. 12, No. 12, December 1979, pp. 9-19.

Shneiderman, B. "Improving the Human Factors Aspect of Database Interactions", ACM Trans. on Database Systems Vol. 3, 1978, pp. 417-439.

Sibley, E.E. and Kershberg, L. "Data Architecture and Data Model Considerations", Proc. AFIPS NCC, Dallas TX, June 1977, pp. 85-96.

Sibley, E.H. and Hardgrave, W.T. "Data Model Theory and Positional Set Processing", in Data Models and Database Systems, Proc. Joint U.S.-U.S.S.R. Seminar (eds. Dale, A.G. and Suvorov, B.P.), Institute for Computer Science and Computer Applications, The University of Texas at Austin, Austin TX, 1977.

Sibley, E.H. and Taylor, R.W. "A Data Definition and Mapping Language", Comm. of the ACM Vol. 16, 1973, pp. 750-759.

Simmons, R.F. "Semantic Networks: Their Computation and Use for Understanding English Sentences", in Computer Models of Thought and Language, (eds. Schank and Colby), San Francisco CA, Freeman, 1973.

Smith, D.C.P. "An Approach to Data Description and Conversion", Ph.D Thesis, Moore School Report 72-20, University of Pennsylvania, Philadelphia.

Smith, J. M. and Smith, D. C. P. "Database Abstractions : Aggregation", Comm. of the ACM, Vol. 20, No. 6, pp. 405 - 413, June 1977.

Smith, J.M. "A Normal Form For Abstract Data Syntax", in Proc. 4th Conf. on Very Large Databases, West Berlin, 1978, pp. 156-162.

Smith, J.M. and Smith, D. C. P. "Principles of Conceptual Database Design", Proc. of NYU Symposium on Database Design, New York NY, May 1978.

Smith, J.M. and Smith, D.C.P. "A Database Approach to Software Specification", Tech. Report CCA-79-17, Computer Corporation of America, Cambridge Mass., April 1979.

Smith, J.M. and Smith, D.C.P. "Database Abstractions: Aggregation and Generalization", ACM Transactions on Database Systems, Vol. 2, No. 2, June 1977, pp. 105 - 133.

Solvberg, A. "A Contribution to the Definition of Concepts for Expressing Users' Information System Requirements", Proc. of Int. Conf. on the Entity-Relationship Approach to Systems Analysis and Design, Los Angeles CA, December 1979.

Sowa, J. "Conceptual Graphs for a Database Interface", IBM Jnl. of Res. and Development, Vol. 20, No. 4, July 1976.

Sperry Univac, "Data Management System (DMS 1100)", UP 7909, Rev 3A, St. Paul Minn., 1972.

Sperry Univac, "Univac Data Management System-1100", Books 1 and 2, Sperry Univac Education Centre, UE 745, St. Paul Minn. (undated.)

Sterling, T.D. "Guidelines for Humanizing Computerized Information Systems: A Report from Stanley House", in Comm. of the ACM Vol. 17, No. 11, November 1974.

Stonebraker, M. R. "High Level Integrity Assurance in Relational Database Mangement Systems.", Electronics Res. Laboratory Report ERL-M473, University of California, Berkeley CA, August 1974.

Su, S.U., Lam, H. and Lo D. H. "Tranformation of Data Traversals and Operations in Application Programs to Account for Semantic Changes of Databases", ACM Trans. on Database Systems Vol. 6, No. 2, June 1981.

Su, S.Y. and Lo, D. H. "A Semantic Association Model for Conceptual Database Design", Proc. of Int. Conf. on the Entity-Relationship Approach to Systems Analysis and Design, Los Angeles CA, December, 1979.

Sundgren, B. "Conceptual Foundation of the Infological Approach to Data Bases", in Data Base Management (eds. Klimbie, J.W. and Koffeman, K.L.), North-Holland, Amsterdam, 1974, pp. 61-96.

Sundgren, B. "Data Base Design in Theory and Practice: Towards an Integrated Methodology", Proc. 4th Int. Conf. on Very Large Databases, 1978, pp. 3-16.

Sundgren, B. "Theory of Data Bases", Mason-Charter, New York NY, 1978.

Taggart, W.M.Jr. and Tharp, M.O. "A Survey of Information Requirements Analysis Techiques", ACM Computing Surveys Vol. 9, 1977, pp. 273-290.

Taylor, R.W. "Man-Computer Input-Output Techniques", IEEE TRans. Hum. Factors Electron., Vol. HFE-8, pp. 1-4, March 1967.

Taylor, R.W. and Frank, R.L. "Codasyl Database Management Systems", ACM Computing Surveys, Vol. 8, No. 1, March 1976, pp. 67-104.

Teichroew, D. and Hershey, E.A. "PSL/PSA: A Computer-Aided Technique for Structured Documentation and Analysis of Information Processing Systems", IEEE Trans. on Software Engineering, Vol. SE-3, 1977, pp. 41-48.

Teorey, T.J. and Fry, J.P. "The Logical Record Acccess Approach to Database Design", Proc. of Int. Conf. on the Entity-Relationship Approach to Systems Analysis and Design, Los Angeles CA, December, 1979.

Thomas, R.E.and Pritsker, A.A.B. "Decision Programming: A Model of Man-Machine Control", IEEE Trans. Hum. Factors Electromn., Vol. HFE-3, pp.25-28, March 1962.

Thompson, D.A. "Man-Computer Systems: Towards Balanced Cooperation in Intellectual Activities", Proc. Int. Symposium Man-Machine Systems, Vol. 1, September 1969.

Tompa, F.W. "Choosing a Data Storage Schema", Ph.D. Thesis, Department of Computer Science, University of Toronto, 1974.

Tompa, F.W. "Choosing an efficient Internal Schema", in Systems for Large Databases, (eds. Lockemann and Neuhold), North-Holland, 1976, Proc. of the 2nd Conf. on Very Large Databases, Belgium, 1976.

Trauboth, H. "Database Design in Software Systems for Process Monitoring and Control", in Database Management (eds. Weber,H. and Wasserman, A.I.), North-Holland, Proc. 4th Conf. on Very Large Databases, 1978.

Treu, S. "Interactive Command Language Design Based on Required

Mental Work", Int. Jnl. of Man-Machine Studies, Vol. 7, 1975, pp. 135-149.

Tsichritzis, D.C. "Features of a Conceptual Schema", Technical Report CSRG-56, Comput. Syst. Res. Group, University of Toronto, Toronto, 1975.

Tsichritzis, D.C. and Klug, A. "The ANSI/X3/SPARC DBMS Framework Report of the Study Group on Database Management Systems", Inf. Syst. Vol. 3, 1978, pp. 173-191.

Tsichritzis, D.C. and Lochovsky, F.H. "Designing the Data Base", Datamation, Vol 24, No. 8, 1978, pp. 147-151.

Ullman, J.D. "Principles of Database Systems", Computer Science Press, Potomac MD, 1980.

Uttal, W. R. "Real-Time Computers: Technique and Applications in the Psychological Sciences", Harper and Row Publishers, New York NY, 1968.

Vassiliou, Y. "Funtional Dependencies and Incomplete Information", Proc. 6th Int. Conf. on Very Large Databases, 1980, pp. 260-269.

Vetter, M. "Database Design by Applied Data Synthesis", Proc. 3rd Int. Conf. on Very Large Databases, October 1977, pp. 428-440.

Walch, G. "Dialogue Programming with Easy Transition from Prompting to Command Mode", IBM Scient. Centre,Technical Report, Heidelberg.

Walker, W.A. "Experience in Data Modelling at Ontario Hydro", Guide 52, Atlanta, 1981.

Walther, G. H. and O'Neil, H. F. Jr. "On-line User-Computer Interface - the Effect of Interface Flexibility, Terminal Type and Experience on Performance", AFIPS Conf. Proc., Vol. 43, 1974 NCC, pp. 379 384.

Waltz, D. "An English-Language Question Answering System", Comm. of the ACM, Vol. 21, No. 7, July 1978, pp. 526 - 539.

Wang, C. and Wedekind, H. "Segment Synthesis in Logical Data Base Design", IBM Jnl. of Res. and Development Vol. 19, January 1975, pp. 71-77.

Wasserman, A. I. "The Data Mangement Facilities of Plain", Proc. of the ACM SIGMOD Int. Conf. on the Management of Data, Boston MA, June, 1979.

Wasserman, A. I. "The Design of Idiot-Proof Interactive Systems", AFIPS Conf. Proc., Vol. 42, 1973 NCC, AFIPS Press, Montvale N.J., 1973, pp. M34-M38.

Wasserman, A.I. "A Software Engineering Approach to Database Management", in Database Management (eds. Weber,H. and Wasserman, A.I.), North-Holland, Proc. 4th Conf. on Very Large Databases, 1978.

Weizenbaum, J. "Contextual Understanding by Computers", Comm. of the ACM, Vol. 10, pp. 474-480, August 1967.

Weizenbaum, J. "ELIZA - A Computer Program for the Sudy of Natural Language Communication Between Man and Machine", Comm. of the ACM, Vol. (, pp. 36-45, January 1966.

Wiederhold, G. , Database Design, McGraw Hill, 1977.

Wiederhold,G. and El-Masri, R. "Structural Model for Database Design, Proc. of Int. Conf. on the Entity-Relationship Approach to Systems Analysis and Design, Los Angeles CA, December 1979.

Winograd, T. "Frame Representation and the Declarative-Procedural Controversy", in Representation and Understanding, (ed.s Bobrow and Collins), Academic Press New York NY, 1975.

Winograd, T. "Understanding Natural Language", Academic Press New York NY, 1972.

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, 1977, pp. 344-383.

Woods, W. A. "What's in a Link: Foundations for Semantic Networks", in Representation and Understanding (eds. Bobrow, D.G. and Collins, A.M.), Academic Press, 1975.

Yao, S.B. "Modelling and Performance Evaluation of Physical Database Structures", ACM Conf. Proc. 1976, pp. 303-309.

Yao, S.B., Navathe, S.B. and Weldon, J.-L. "An Integrated Approach to Logical Database Design", Proc. NYU Symposium on Database Design, 1978, pp. 1-14.

Yntema, D. B. "Engineering Psychology in Man-Computer Interaction", Paper presented at Annual Convention of the American Psychological Association, San Francisco CA, 1968.

Zaniolo, C. "A New Normal Form for the Design of Relational Database Schemata", ACM Trans. on Database Systems, Vol. 7, No. 3, September 1982, pp. 489-499.

Zaniolo, C. and Melkanoff, M.A. "A Formal Approach to the Definition and the Design of Conceptual Schemata for Database Systems", ACM Trans. on Database Systems, Vol. 7, No. 1, March 1982, pp. 24-59.

Zaniolo, C. and Melkanoff,M.A. "On the Design of Relational Database Schemata", ACM Trans. on Database Systems Vol. 6, 1981, pp. 1-47.

Zeigler, B.P. and Sheridan, T.B."Human use of short-term memory in Processing Information on a Console", IEEE Trans. Hum. Factors Electorn., Vol. HFE-6, pp. 74-83, September 1965.