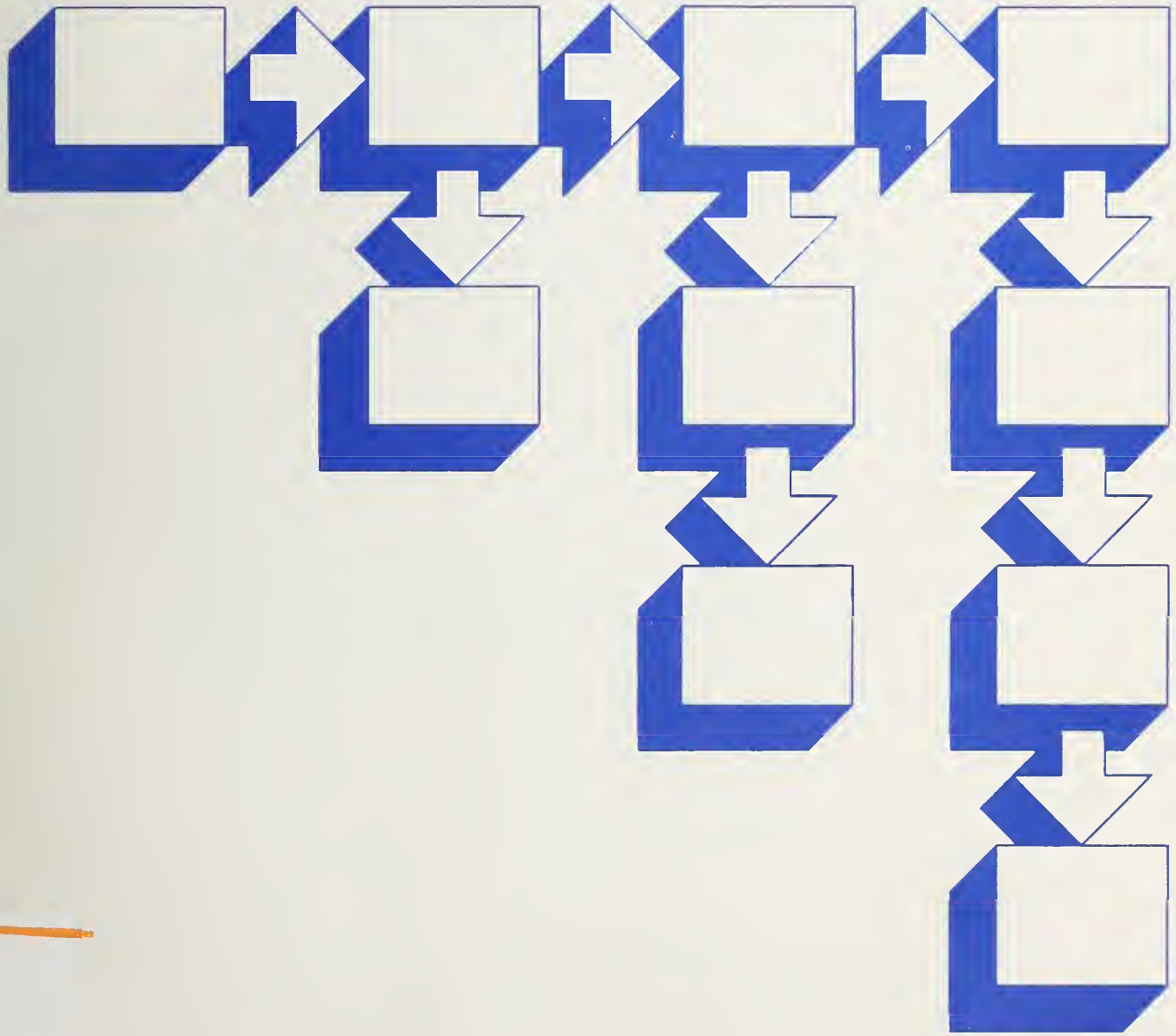
U.S. Department of Commerce
National Bureau of Standards

Computer Science and Technology

NBS Special Publication 500-108

Guide on Data Models in the Selection and Use of Database Management Systems



NATIONAL BUREAU OF STANDARDS

The National Bureau of Standards[1] was established by an act of Congress on March 3, 1901. The Bureau's overall goal is to strengthen and advance the Nation's science and technology and facilitate their effective application for public benefit. To this end, the Bureau conducts research and provides: (1) a basis for the Nation's physical measurement system, (2) scientific and technological services for industry and government, (3) a technical basis for equity in trade, and (4) technical services to promote public safety. The Bureau's technical work is performed by the National Measurement Laboratory, the National Engineering Laboratory, and the Institute for Computer Sciences and Technology.

THE NATIONAL MEASUREMENT LABORATORY provides the national system of physical and chemical and materials measurement; coordinates the system with measurement systems of other nations and furnishes essential services leading to accurate and uniform physical and chemical measurement throughout the Nation's scientific community, industry, and commerce; conducts materials research leading to improved methods of measurement, standards, and data on the properties of materials needed by industry, commerce, educational institutions, and Government; provides advisory and research services to other Government agencies; develops, produces, and distributes Standard Reference Materials; and provides calibration services. The Laboratory consists of the following centers:

Absolute Physical Quantities[2] — Radiation Research — Chemical Physics — Analytical Chemistry — Materials Science

THE NATIONAL ENGINEERING LABORATORY provides technology and technical services to the public and private sectors to address national needs and to solve national problems; conducts research in engineering and applied science in support of these efforts; builds and maintains competence in the necessary disciplines required to carry out this research and technical service; develops engineering data and measurement capabilities; provides engineering measurement traceability services; develops test methods and proposes engineering standards and code changes; develops and proposes new engineering practices; and develops and improves mechanisms to transfer results of its research to the ultimate user. The Laboratory consists of the following centers:

Applied Mathematics — Electronics and Electrical Engineering[2] — Manufacturing Engineering — Building Technology — Fire Research — Chemical Engineering[2]

THE INSTITUTE FOR COMPUTER SCIENCES AND TECHNOLOGY conducts research and provides scientific and technical services to aid Federal agencies in the selection, acquisition, application, and use of computer technology to improve effectiveness and economy in Government operations in accordance with Public Law 89-306 (40 U.S.C. 759), relevant Executive Orders, and other directives; carries out this mission by managing the Federal Information Processing Standards Program, developing Federal ADP standards guidelines, and managing Federal participation in ADP voluntary standardization activities; provides scientific and technological advisory services and assistance to Federal agencies; and provides the technical foundation for computer-related policies of the Federal Government. The Institute consists of the following centers:

Programming Science and Technology — Computer Systems Engineering.

[1]Headquarters and Laboratories at Gaithersburg, MD, unless otherwise noted; mailing address Washington, DC 20234.
[2]Some divisions within the center are located at Boulder, CO 80303.

Computer Science and Technology

NBS Special Publication 500-108

Guide on Data Models in the Selection and Use of Database Management Systems

Leonard J. Gallagher
Jesse M. Draper

Center for Programming Science and Technology
Institute for Computer Sciences and Technology
National Bureau of Standards
Washington, D.C. 20234

U.S. DEPARTMENT OF COMMERCE
Malcolm Baldrige, Secretary

National Bureau of Standards
Ernest Ambler, Director

Issued January 1984

Reports on Computer Science and Technology

The National Bureau of Standards has a special responsibility within the Federal Government for computer science and technology activities. The programs of the NBS Institute for Computer Sciences and Technology are designed to provide ADP standards, guidelines, and technical advisory services to improve the effectiveness of computer utilization in the Federal sector, and to perform appropriate research and development efforts as foundation for such activities and programs. This publication series will report these NBS efforts to the Federal computer community as well as to interested specialists in the academic and private sectors. Those wishing to receive notices of publications in this series should complete and return the form at the end of this publication.

Library of Congress Catalog Card Number: 83:600630

National Bureau of Standards Special Publication 500-108
Natl. Bur. Stand. (U.S.), Spec. Publ. 500-108, 71 pages (Jan. 1984)
CODEN: XNBSAV

U.S. GOVERNMENT PRINTING OFFICE
WASHINGTON: 1984

For sale by the Superintendent of Documents, U.S. Government Printing Office, Washington, DC 20402

TABLE OF CONTENTS

Guide on Data Models in the Selection and Use of Database Management Systems

Leonard J. Gallagher
Jesse M. Draper

Selecting a database management system involves matching users' requirements and the capabilities of available products. One way to simplify this task is to define data models identifying both data structures and the operations on those structures. In the past every commercial product has implemented its own data model. Now technical committee X3H2 of the American National Standards Institute is working on specifications for two models that are similar but not identical to many existing products. The network model is a structure-oriented model that is especially suitable for databases with static structures and a high volume of record-at-a-time processing. The relational model depends more heavily on operations than structures and thus provides the flexibility to handle dynamic databases. Examples written in the draft Network Database Language and the Relational Database Language demonstrate that both models can answer complex queries in a straightforward manner.

In addition to the issue of data models, prospective buyers of database software need to consider features that affect daily operations. Existing hardware and operating systems sometimes limit the choice to a few commercial products. Systems also vary widely in their facilities for backup and recovery, bulk loading, schema manipulation, concurrency control, and report writers.

Key words: computer languages; computer software standards; DBMS; data management; data models; database management systems; network databases; relational databases; system selection.

1. INTRODUCTION

Since the early 1960's, when the formatted file systems that preceded modern generalized database management systems (DBMS's) were first introduced, application developers have been able to select from an increasing number of research and commercial DBMS products. Recent articles in the trade press list well over one hundred different vendors marketing nearly two hundred separate database management products [PERS81, SOFT82]. New products are announced continuously, especially for small and medium-sized computers.

Today -- without any international, national, or Federal standards -- virtually every commercial database product is unique. Furthermore, extant database management systems are described primarily by reference documents that are sometimes incomplete: the products themselves provide the ultimate specifications. It is difficult, therefore, to characterize classes of these nonhomogeneous and undocumented database management systems. Potential users need a method for differentiating these products according to their fundamental capabilities without becoming overwhelmed by highly specialized features of specific implementations. The concept of a data model provides such a means for classifying and understanding implementations of database management systems. Fortunately, many DBMS products are based upon one of a small number of data models that have received extensive attention in the research literature. Two of these models -- the network and the relational -- are current candidates for American national standards.

This report identifies the characteristics of several major data models, with special emphasis on the two proposed standards. No one data model is uniquely appropriate for all database applications; the special characteristics of each application will determine the most appropriate data model.

1.1 Data Models and Database Management Systems

A data model is a collection of data structures together with a collection of operations that manipulate the data structures for the purpose of storing, querying, or processing the structure contents. A data model may also include the integrity constraints defined over the data structures, or it may include access control facilities or mechanisms for defining various external user views of the database. Some data models provide physical storage structures and

physical access methods as part of the data model, but usually a data model is limited to the data structures and operations that are available to an end user and may be accessed from an application program.

A database management system is a general purpose, application independent, software package used in association with computer hardware to facilitate the entry, storage, processing, sharing, and retrieval of data from a database. The portion of a DBMS that deals directly with the processing of the data structures of a data model is sometimes referred to as the database control system. A DBMS supports a data model and is an implementation of that data model. Some database management systems may support multiple data models by providing different user interfaces to the database. A DBMS provides for transformation of the logical data structures of a data model to the physical storage structures of a particular hardware environment. The DBMS goes beyond the data model in that it must provide for communication with the operating system of the host computer, as well as interface with programming languages or associated software systems such as data dictionaries, report writers, statistical packages, graphics, and libraries of special data processing functions. A DBMS generally provides concurrency control, backup, and restart, as well as dumping and loading facilities for all databases under its control.

1.2 Evolution of Data Models and Standards

In the early days of data processing, when external storage consisted of punched cards or paper or magnetic tape, all file access was sequential; no data could be retrieved without first passing over all previously stored data. Even with this restriction, however, data processing thrived since many applications like payroll or invoicing required no more than this limited access method. During this time a data model consisted of just a sequence of records, although many sophisticated specifications of master and trailer records were forerunners of more general structures in later data models.

With the advent of drum and disk external storage came the notion of direct record access where each record carried a unique record identifier that could be used in indexes or in chains of related records. This led, in the mid 1960's, to early hierarchical data models that allowed direct access of master records and record-to-record navigation over subordinate trailer records. It was also during this period that significant strides were made in shared access to data.

No longer was data owned by a single application; instead, data files were stored in a "database" separate from application programs, with file access controlled by a "database management system."

The late 1960's and early 1970's brought a flurry of database research. Charles W. Bachman is widely recognized as one of the early developers of the network approach to data management; his 1964 paper with S.B. Williams presented a flexible scheme for linking together records of different types using a pointer-chain structure [BACH64]. E.F. Codd wrote the 1970 seminal paper that defined the concepts of normalization and joins for relational tables [CODD70]. This paper created a good deal of interest in various high level query and manipulation languages based on a predicate calculus. By the late 1970's both Quel [STON76] and Sequel [ASTR75] had achieved popularity as very powerful yet user friendly data manipulation languages. The hierarchical approach to database management derives from a generalization of the repeating group structures found in many programming languages. Hierarchical systems evolved independently in the 1960's so that currently there are many different versions in the marketplace. Finally, The entity-relationship approach for describing a database became popular in 1976 with the publication of an article by P.P. Chen [CHEN76]. The model proposed originally did not include specification of any data manipulation operations; instead, it focused on the specification of entity types and the relationships among them. Later authors have specified operations to make it a complete data model [JOHN82, SHIP81]. Two conferences have been devoted to the logical design and application of databases defined using this approach [CHEN79, CHEN81].

The first attempt at a standard specification for a specific data model occurred during 1967-71 when the CODASYL Data Base Task Group defined structures and operations for a network database facility in the COBOL language. The first ANSI activity leading to recognition of different data models occurred in 1977 with completion of the ANSI/X3/SPARC Data Base Study Group report that presents a three-schema architecture for database management [SPAR77]. An ANSI technical committee, X3H2, was established in 1978 to define a data definition language for interface from various data manipulation languages defined by the programming language committees. In 1981, the scope of X3H2 expanded to include definition of ANSI standard structures and operations for the network data model. Finally, in 1982, X3H2 was asked to develop similar definitions for the relational data model. The results of these tasks are the Network Database Language (NDL) and the Relational Database Language (RDL), two draft ANSI specifications that are discussed in this report.

These specifications are now fairly stable, but they may change before final adoption.

As mentioned earlier, there are no existing database standards at either the international, national, or Federal levels. However, the ANSI proposals are under critical review by a special international database experts group that will make recommendations to its parent International Standards Organization committee. Federal representatives have been active participants in all ANSI database committees. It appears likely, at least in the near term, that international and Federal standards will derive from and be consistent with resulting ANSI standards.

An important feature of the planned database standards is that no single interface specification will exclude other interfaces between the end user and the database. For example, both the NDL and RDL assume a programming language interface to the data, yet they acknowledge the existence of other user interfaces such as: ad hoc query and report writer languages, schema manipulation languages or data dictionary interfaces, special transaction processing systems that take advantage of modern screen and graphics capabilities, and bulk loading or unloading facilities both for database backup and for database information interchange. Language specifications for these additional capabilities are, at present, unique to each DBMS vendor. If and when standard specifications become available, they should be upwardly compatible with established data model standards.

1.3 Purpose and Organization of this Report

The purpose of this report is to provide a tutorial introduction to data models in general, with particular emphasis on the relational and network models defined by the two proposed ANSI database language standards. Even though no current commercial product satisfies either specification exactly, the specified structures and operations are typical of existing capabilities in a wide variety of DBMS products. Thus the proposed languages can be used for comparison purposes in DBMS selections made even before the existence of conforming products.

The next three chapters of this report constitute the tutorial introduction to data models. Examples based on the network and relational models [X3H283a, X3H283b] include specific syntax and semantics, whereas examples from the other models are necessarily less precise. Chapter 2 describes an example database and then focuses on database

structures and their definitions in several models. Data manipulation is the topic of Chapter 3, with a discussion of access from external languages and a number of examples designed to show the power of the network and relational models in manipulating their data structures. Because examples alone cannot demonstrate all the features of a language, Chapter 4 discusses some characteristic features of these two data models -- both their particular benefits and their limitations.

While choosing the right data model for a database is probably the most important aspect of DBMS selection, the data model per se does not specify all the essential features of a product. Chapter 5 discusses many of the other issues in the selection of a DBMS: access control, back-up, recovery, bulk loading, and concurrency control. This chapter is not intended to be used as a specification; instead, it is included to aid in recognition of critical issues in the selection process. Most of the topics mentioned in Chapter 5 pertain to the daily operation of the DBMS itself rather than to the logical structures and operations of the data model.

2. DATA STRUCTURES AND DEFINITION

For the purposes of this tutorial the examples refer to a database of information about employees and departments. Each department has a unique name, a specific location, and a set of employees. Employee records include name, age, manager, and salary, plus a history of the employee's positions in the organization. Users of the database must be able to retrieve employee information either directly or by department. In describing the structures for modeling this database, the following paragraphs focus on data types, the actual structures, integrity constraints, and schema definition.

2.1 Data Types

Every data model has a particular collection of data types. A data type is the definition of a set of values that can be represented in a data model. A value is primitive: it has no logical subdivision within the data model.

Such primitive values are the basis of definition for the other data structures of the model. Most models have a character string data type and at least one numeric data type. Some make a distinction between fixed-length and variable-length strings, or between exact and approximate numeric values. Some numeric types may be defined with different degrees of numeric precision. Other common data types include calendar date, time-of-day, zip code, sex code, Boolean, money, complex numbers, long strings of text, enumeration types, or various forms of pointers and identifiers.

The ANSI database committee has defined three common data types for network and relational databases. They include character strings, exact or fixed point numbers, and approximate or floating point numbers. In addition, the proposed NDL and RDL specifications accommodate corresponding data types in various ANSI standard programming languages (e.g., FORTRAN, COBOL, PL/1, and Pascal). Instead of trying to describe the specification of each language-specific data type in the database interface, we will focus on the common types in our examples, referring to them as CHARACTER, FIXED, and FLOAT, respectively.

A character string is a finite sequence of characters taken from some well-defined character set (e.g., ASCII, EBCDIC, BCD). Character sets for databases may include: the human readable "graphic characters" as specified by ANSI X3.4 [ANSI77], the complete set of 128 characters as specified by ANSI X3.4, various international character sets as specified by ISO standards [ISO73, ISO82] or vendor specific character sets. Each character string has a fixed length, a positive integer associated with the string that describes the number of characters in that string. Strings may be of variable length up to the fixed length, but logically they are padded with blank characters when used in comparisons.

Numbers are values that have normal mathematical properties; they are defined as real numbers with decimal base. Fixed point numbers are assumed to be exact values, with an associated precision and scale factor. The precision specifies the number of significant decimal digits, and the scale factor specifies the placement of the decimal point. Floating point numbers, which are assumed to be approximate values, consist of a significand and an exrad. The significand is a fixed point number, and the exrad is an integer. The value of the number is the value of its significand multiplied by 10 to the power of the exrad. Every floating point number has a precision that specifies the precision of the significand.

2.2 Structures

While data types should be consistent from one data model to another, data structures will by definition vary. What distinguishes a data model is not the lowest level values, but the organization of those values into structures and the provision of appropriate operations on those structures. The following sections show what a variety of data structures can be built up from the elementary data types just described.

2.2.1 The Network Model. The network data model contains two basic data structures: records and sets. As the basic units of data manipulation, records are stored, erased, found, modified, and connected and disconnected from other records. Sets, the basic units of navigation, maintain inter-record relationships. Using logical set access paths defined by the database schema, a user can move from one record to another.

A record is a collection of data components, each of which is either a data item or an array. A data item consists of a single value; an array is a multi-dimensional table of values that is represented by a sequence of data items. Each such array has a fixed dimension that is a positive integer. Every positive integer less than or equal to the dimension determines a direction for that array, and each direction has an extent that is also a positive integer. For example, in a two-dimensional array there are two directions. The extent in the first direction determines the number of rows of the table, and the extent in the second direction determines the number of columns. The number of data items that occur in an array is the product of the extent integers of that array. A data item within an array is referenced by a multi-dimensional subscript. An implicit row-major ordering of array items establishes a unique correspondence between a data item referenced by a subscript and its sequential position in the array representation. For example, in a two-dimensional array with three rows and four columns, items 1-4 occupy the first row, items 5-8 the second row, and items 9-12 the third row. The subscript (2,3) thus references the seventh sequential position.

All records in a network database are partitioned according to record type. A record type defines the components of each record occurrence of the record type and declares a record name for the record type and a component name for each component. Each record of the database is an occurrence of exactly one record type and consists of

exactly the data items defined by that record type.

A set is a structured collection of related records. It models the classical data processing notion of master and trailer records. Each set is an occurrence of a set type, which is the definition of a collection of sets all having the same characteristics. The declaration of a set type specifies the name of the set type and the owner and member record types that are associated with the set type. A set establishes a relationship among its component records that must be maintained by the DBMS. One record from each set is designated as the owner record of that set. Any other record in the set is a member record. Given a set type, there is exactly one set for each record of the owner record type. That is, in practice an occurrence of the owner record determines an occurrence of the set. Each set may contain zero or more occurrences of each member record type, but each member record occurrence may belong to at most one set. In practice, this restriction allows navigation from a member record to its unique owner record. The member records of each set of a set type are maintained in a sequential order determined by the ordering criteria of that set type.

The network model supports two special set types: singular and recursive. A singular set type has SYSTEM declared as its owner record type. SYSTEM can be thought of as a special record type containing exactly one record with no data items. Hence, there is only one occurrence of a singular set type, and it allows direct access to member records without first navigating through multiple occurrences of an owner record type. A recursive set type has the same record type declared both as the owner record type and as a member record type; thus, it allows convenient representation of hierarchical relationships, such as manager/employee, among records of the same record type.

All records must be distinguishable by the DBMS, including those that participate in the same data structures and have the same values for each component data item. For this reason, each record is associated with a unique record identifier called a database key, which is a conceptual, implementation-dependent object used to maintain position in the database. Database keys are not directly available to an end user. The action initiated by any database statement is dependent upon the database keys that occur as values of special cursors maintained by the DBMS in a session state for each database session.

One convenient network structure for the sample database is illustrated in Figure 1. This view of the database is defined by a schema named COMPANY. Each rectangle represents a record type with the type name in the top half and the component names in the lower half. Set types appear as labeled arrows drawn from the owner record type to the member record types. The oval labeled SYSTEM represents the owner of a "singular set" that, according to a schema order clause, provides an alphabetical ordering of member records by employee name.

In this example the DEPT record type contains components named NAME and LOC to contain the department's name and location. Each department's employees form an occurrence of the set type PAYROLL, which, through an order clause in the schema, orders the employees of each set according to decreasing salary. Some of the information about an employee -- name, age, and salary -- appears explicitly in the EMP record identified by the employee's name. Other information, like the employee's department and manager, is actually contained in the structure of the database rather than in a component within a given record. For some applications it might make more sense to include in each employee record one component containing the department and another naming the manager. That, too, would be a legitimate network structure, but it would not illustrate as many features of the network data model. The recursive set type MANAGES determines a one-to-many relationship from a manager to the employees directly supervised by that manager. Each employee is the owner of a MANAGES set occurrence; however, if that person is not a manager, the set has no member records. Note that one restriction inherent in the set structure is that no employee has more than one manager. Finally, the database contains in the record types SUPERVISORS and STAFF the employment history of each employee. Notice that the only specified ordering of records of these types is through their membership in sets of the type JOBHISTORY. And since neither of these types includes a name component, their records are connected to an employee only by membership in a particular JOBHISTORY set, not by redundant data values.

2.2.2 The Relational Model. In the relational model the primary data structure is the table, which is defined as an unordered collection of rows that are not necessarily distinct. This assumption of nonuniqueness for rows contradicts some theoretical definitions of the relational model, but it is consistent with most current implementations and with many user requirements. Uniqueness is enforced, if necessary, by an integrity constraint. A row, which is the smallest unit of data that can be stored into or erased from

Network Data Structures

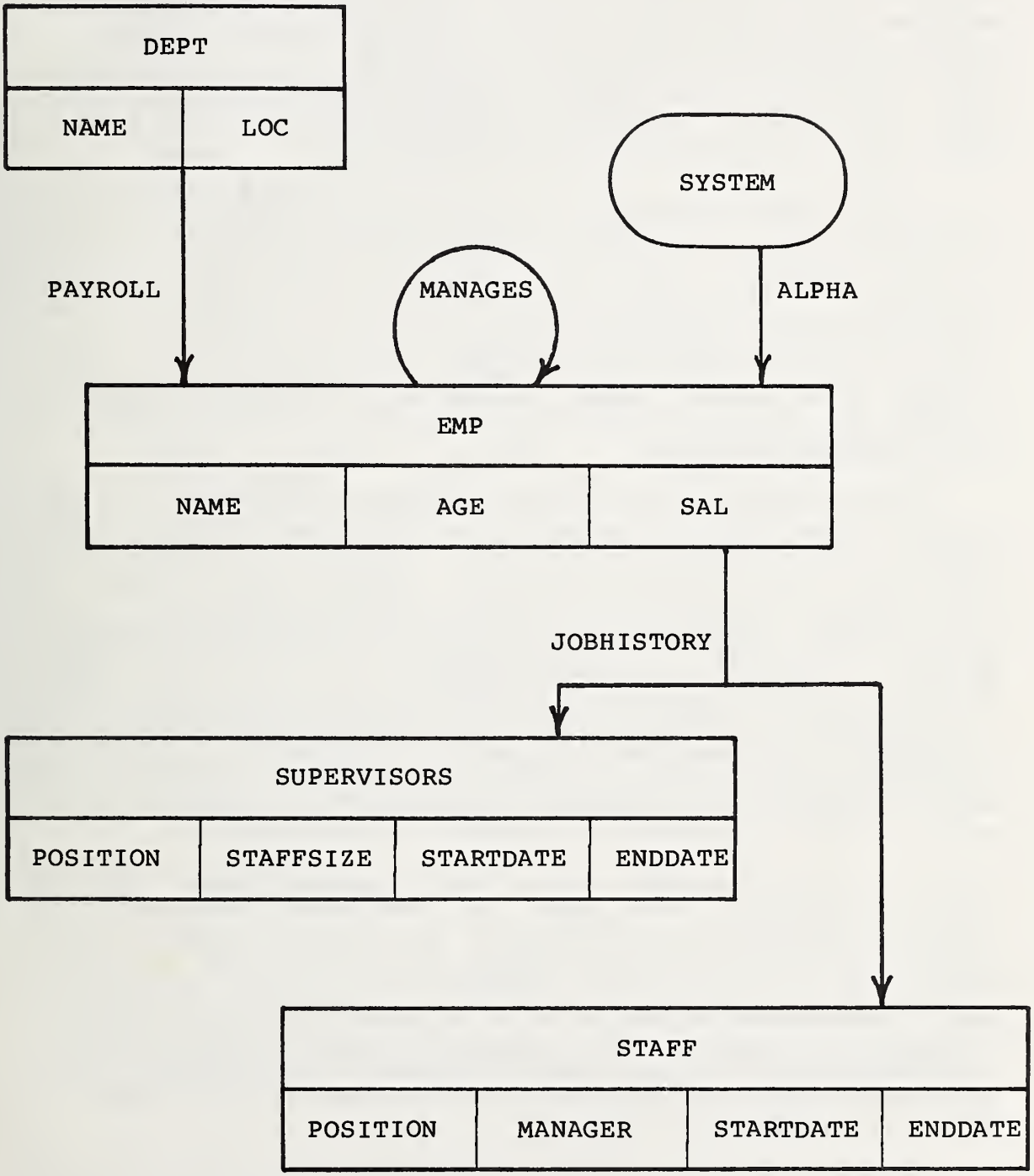


Figure 1

a table, is a non-empty sequence of values. The assumption that the values in a row are ordered is only important when values are referenced by position rather than by name. Each value belongs to a column, which is an unordered, named collection of values of the same elementary data type. The column is equivalent to the concept of a domain included in some theoretical definitions. A column entry is the smallest unit of data that may be selected from a table or modified in a table. Unlike the network model, the relational model does not support arrays of values.

Each table is associated with a table definition that specifies the table name and table characteristics as well as the column name and column characteristics of each column of that table. Every row of the same table has the same cardinality and contains a value for every column of that table. The relational model supports the notion of a "null" value, which is comparable with but distinct from all other values. This special value is assumed for a column position in a row whenever no default or other non-null value is specifically assigned.

Tables may be base tables, derived tables, or viewed tables. Base tables have persistent storage representations and persistent table descriptions, similar to the record types and set types defined by a network model schema. A derived table is a temporary table derived from one or more base tables during the execution of a database statement. As such, it does not become a permanent part of the database; it exists no longer than does the transaction in which it is defined. Viewed tables are derived tables that have persistent descriptions. Each viewed table provides subschema views of the database to external users. All three types of tables may be the objects of database statements in the relational data manipulation language.

All rows in a table must be distinguishable by the DBMS, including those rows in the same table that have identical values for each column. For this reason, relational implementations must rely on some kind of unique row identifier, be it physical location in a file or use of logical indicators or pointers, to distinguish rows. However, the relational model differs from the network model in that row identifiers are not used to represent inter-record relationships; they are used only to maintain a cursor position within an individual table.

Figure 2 shows a possible relational structure for the sample database. There are four tables: DEPT, EMP, SUPERVISORS, and STAFF. Whereas in the network database some relationships could be based on specifications of set types,

Relational Data Structures

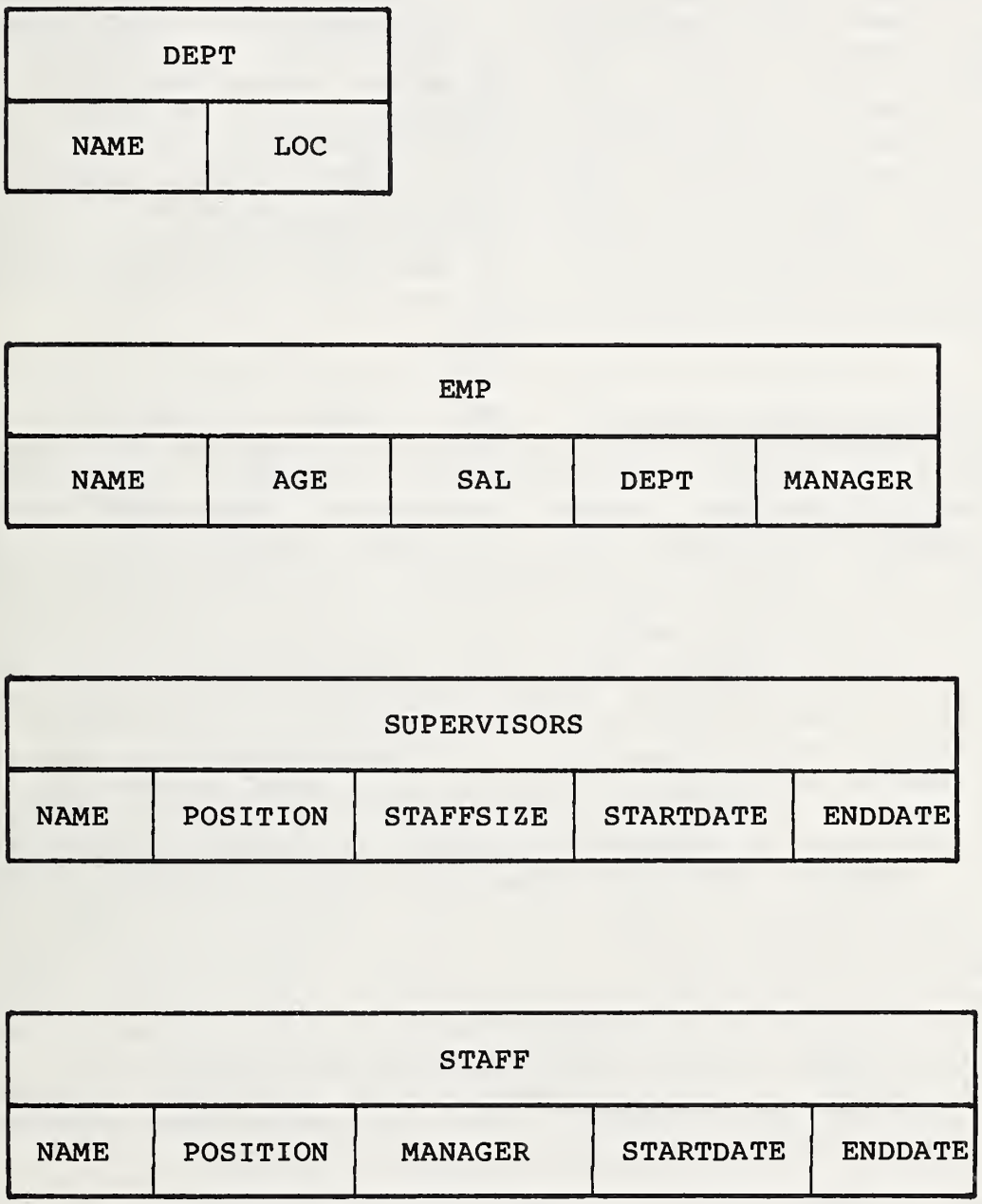


Figure 2

in the relational model they depend on redundant data. Note, for example the MANAGER column in the EMP table, which takes values that must also appear in the NAME column of another row somewhere in the EMP table. In addition, all of the tables except DEPT include a component for the employees' names, and employees and departments are associated by dynamic comparisons between the NAME column in DEPT and the DEPT column in EMP. The DEPT column in the EMP table thus partitions the employees by department in the same way that the PAYROLL sets do in the network example. The relational model does not maintain ordering of rows in base tables, so if the ordering of employees by decreasing salary in the network PAYROLL sets or the alphabetical ordering of employees in the network ALPHA set is important to an application, then ordering criteria would be specified by a cursor declaration in an accessing module.

2.2.3 Hierarchical Models. The basic structures of hierarchical models may be viewed as a subset of the network model data structures defined above. The main structure is a node (sometimes called segment or component) that is essentially equivalent to a network model record type. Nodes are connected one to another in a parent-child relationship very much like the owner to member record relationship in a network model set type. The major restriction is that no child node may have more than one parent node associated with it. In the network model this restriction would prohibit a record type from being a member of more than one set type. The hierarchical model thus places the same restriction on set types that the network model places on set occurrences. This restriction simplifies data definition to the point that it is not necessary to define path names (i.e. network model set names) for the link between parent and child nodes; each such path is uniquely identified once the owner and child nodes are known. Another restriction is that hierarchical models often do not allow either direct access to a child node (as does a network model singular set) or circular node connections as in the network model recursive set. These restrictions on data organization limit the flexibility for defining highly integrated databases, but provide certain capabilities for operational efficiency and retrieval flexibility.

Figure 3 shows a possible hierarchical structure for the sample database. The DEPT node is the "root" of the tree and has one child node (EMP) in addition to its own data items. EMP in turn has two children, the nodes SUPERVISORS and STAFF. In this particular model the relationship among nodes is conceptually tighter than in the network model. Each DEPT record includes a number of EMP records, each of

Hierarchical Data Structures

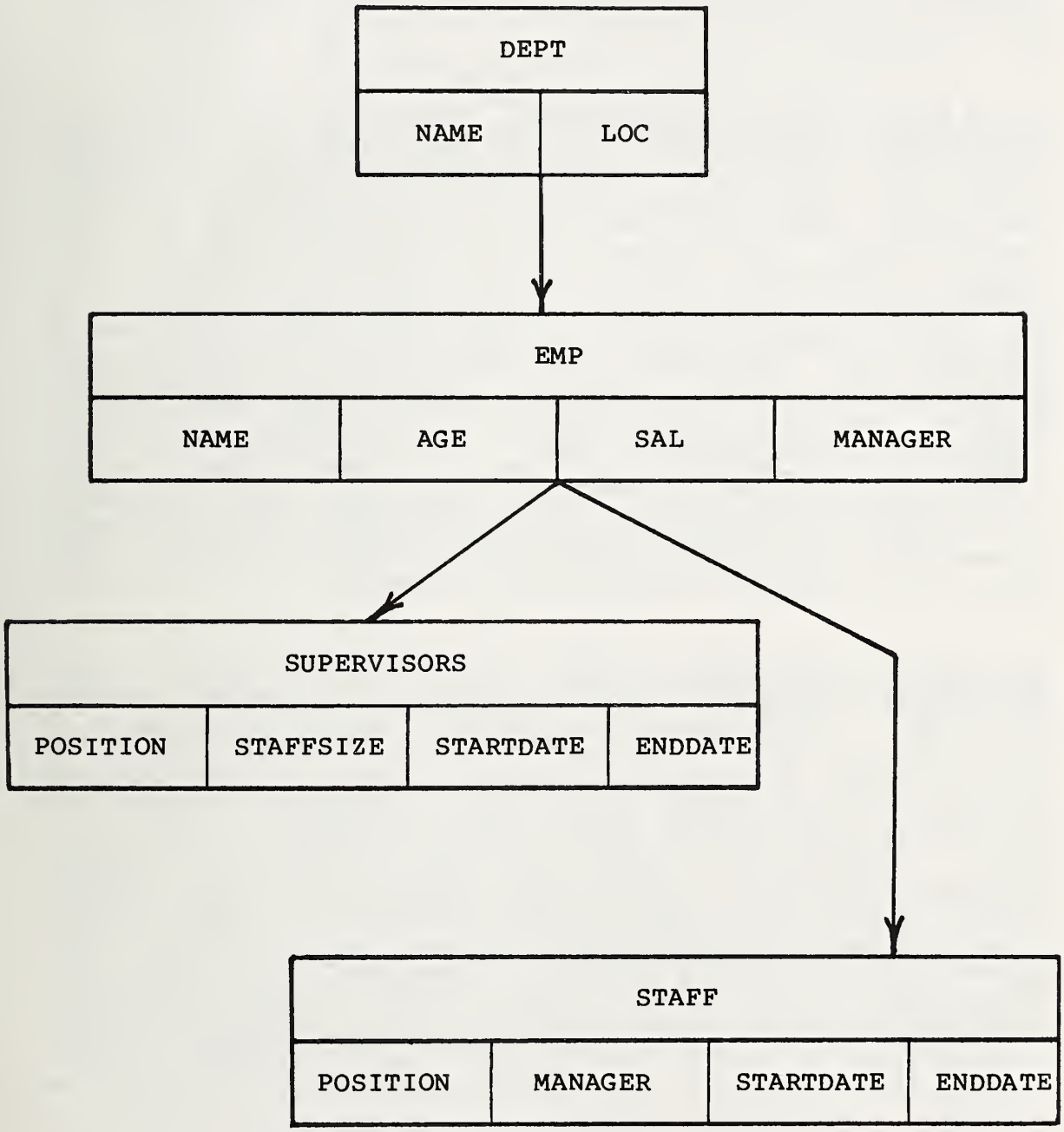


Figure 3

which includes either SUPERVISORS or STAFF records or both. Child node occurrences cannot exist independently of a parent node. The implementation of a hierarchy, however, could be similar to that of a corresponding network database. In either case relationships result from defined structures (perhaps pointers) rather than from redundant data values.

2.2.4 Entity-Relationship Model. The entity structure of this model may be considered as equivalent to a relational table or a network record type. In most cases only elementary data items are considered, so the array structure of the network model would not apply. As in the case of both record types and tables, one must assume the existence of unique entity identifiers to distinguish among entities that have identical data values. In most cases, such identifiers are not externally visible or accessible. The relationship structure of this model is a generalization of the network model set type. Instead of only one-to-many associations between one owner entity type and one or more member entity types, the relationships may be many-to-many among any number of participating entity types. The entity-relationship model is still in its research stages [CHEN81], and commercial implementations have not yet established themselves in the marketplace. The structures are used primarily for logical database design [CHEN82].

2.2.5 Other Structures. Many database management systems provide for the storage and manipulation of variable repeating items. As an example, an employee record may contain a variable repeating item for the names of family members or for a listing of multiple telephone numbers. Early specifications of the CODASYL network model included definition of this capability through nested repeating groups. Since CODASYL specifications derived from a COBOL programming language interface, repeating groups were defined by nested level numbers and OCCURS clauses just as in a COBOL record. A multi-dimensional table was defined using nested level numbers with an OCCURS clause at each level. Such repeating groups were deleted from the ANSI network model specifications because each repeating group is representable in terms of arrays or in terms of additional record types and set types. As an example, consider a nested record structure as follows:

```
RECORD NAME IS  R
 01  A  PIC X(10)
 01  B  OCCURS 5 TIMES
     02  C  PIC X(10)
     02  D  OCCURS 2 TIMES  PIC X(10)
```

This structure could be represented as a single network model record type with A as an elementary item and C and D as arrays. The array C would be one-dimensional, occurring 5 times, and the array D would be two-dimensional, occurring 5 by 2 times. Any references derived from the data name B would be lost to the data model, as would the association between occurrences of C and the first direction of D. Explanations of such associations could be carried along as comments.

A second representation of the above record structure could be through the definition of additional record types and set types. This approach is particularly suitable if data components B and D occur a variable number of times instead of a fixed number. For example, A could be an elementary item in R, C an elementary item in S, and D an elementary item in T, where R, S, and T are different record types. The names of record types S and T would be derived from data components B and D respectively. Record type R would be the owner of a set type with S as a member, and S would be the owner of a set type with T as a member.

The same situation could be described in the relational model using three tables logically associated by the appropriate primary and secondary keys (cf. Section 2.3.2) and referential integrity constraints. For example, the tables R (A,...), S (A,C,...), and T (A,C,D,...) would convey essentially equivalent information with the following restrictions: column A in table R and both column A and column C in table S would have to be declared unique; column A of table S would have to reference column A of table R; and columns A and C of table T would have to reference columns A and C of table S. If the columns defined above were all of the record, then table T alone would suffice. We have added R and S to account for the cases where there are other components at both the 01 and the 02 levels. These components would constitute the other columns in tables R and S, respectively.

The CODASYL specification of the network model also includes definitions for additional structures. An area is a collection of records together with a sequential ordering over them. This structure exists in many commercial products, usually as the logical grouping of record type

populations stored together on the same physical file or storage device. Areas were deleted from the ANSI database definition because each area is logically equivalent to a singular set type with multiple member record types. The equivalence can be demonstrated by defining exactly one additional singular set type to represent each area. The area name becomes the set name of the new set type. Each record type having records in the area becomes a member record type of the singular set type, and the sequential order of records in the area determines the record order of member records in the singular set.

A record order key is the declaration in a record type that record occurrences shall be ordered on given data items in a specified way. Record order keys were deleted from the ANSI network model specification because each such key is logically equivalent to a singular set type defined over that record type. The record order declaration becomes the order declaration for the member records in that set.

2.3 Integrity Constraints

Most data models provide some mechanisms for ensuring the integrity, or validity, of values in the database. These mechanisms can be as simple as a specification of a domain that includes all valid values for a particular component or column, or as complex as an inter-record relationship that must be maintained by every transaction. Integrity constraints depend on data structures and vary considerably from model to model.

2.3.1 The Network Model. A network database definition may include specification of certain integrity constraints on the data items, records, and sets of the database. For example, the length of strings and the precision of numbers are important constraints that should reside with the data itself. Additional integrity constraints may come in several forms. A check condition is an expression that must be satisfied by the values of a record when it is stored in the database or inserted as a member record in a set. A default value is a value assumed by component occurrences in the absence of a specific value supplied by a user, and a unique constraint is a specification that no two records may occur in the database with identical values for specified components. Each set type description also includes specific integrity declarations. A set ordering specifies whether the logical ordering of member records in a set is sorted, first, last, next, prior, or system default. If order is

sorted, then a key declaration specifies the data items that determine the order key. Set insertion declarations specify whether the insertion of a record as a member of a set is automatic, structural, or manual. If insertion is structural, then values for the structural data items determine how an owner record is selected from the database. A set retention declaration specifies whether the retention of a member record in a set is fixed, mandatory, or optional. Examples of these integrity constraints occur in Chapter 4.

2.3.2 The Relational Model. Some implementations or user installations of the relational model include definitions for the primary key and secondary key concepts. A primary key is declared for a single table; it consists of a column whose values uniquely identify a row of the table. A secondary key is also declared for a single table; it relates back to some existing primary key in a different table. The secondary key specifies a column of its table that assumes values comparable to values of its associated primary key. The secondary key value of each row in a secondary key table identifies a unique row in the primary key table. In the absence of specific data structures to represent inter-record relationships, primary and secondary keys are often used to maintain logical connections between tables. In the ANSI X3H2 relational specifications, the notions of primary and secondary keys are not explicitly defined. Instead, a primary key may be assumed whenever a table unique constraint is specified over a single column, and a secondary key may be assumed whenever a table referential constraint is specified. A referential constraint requires that the "secondary key" of each row of the referencing table have a value that is identical to the "primary key" of some row of the referenced table. A referential constraint has some of the same features of the network model set type in that modify and delete statements may cascade from the primary key row to secondary key rows.

A relational database definition may include specification of additional integrity constraints on the rows and columns of tables in the database. As with the network model, this includes specification of length of strings and precision of numbers, as well as declaration of default values for columns and check conditions for rows and columns. Relational table constraints also include the unique and referential constraints defined in the preceding paragraph. Such integrity constraints are often considered to be part of the database structure.

2.3.3 Other Models. The integrity constraints of a hierarchical database are specified differently by each implementation of a hierarchical data model. In most cases the integrity constraints are similar to possible constraints in the proposed network and relational standards. Other models, including the entity-relationship model, can provide similar integrity constraints. In each case there can be atomic constraints on particular values or more elaborate mechanisms (such as triggers) that automatically alter data items which depend on values changed by a user.

2.4 Schema Definition

Every data model must provide some means of defining the structures of a particular database. Some models combine data definition and data manipulation in a single language, but most separate them in one way or another. In some cases data definition is the province of the database administrator alone.

Because the proposed ANSI database languages accommodate language-specific data types, each schema or table definition has a language environment clause identifying an ANSI standard language. To avoid focusing on a single language, we have chosen to use in our examples a pseudolanguage with the types CHARACTER, FIXED, and FLOAT. In an actual implementation PSEUDOLANGUAGE would have to be a standard language listed in the ANSI database specification, and the type declarations would match those of the chosen language. Programs written in other languages could access the database, but either the programs themselves or the language compilers would be responsible for conversions between database data types and the data types of the accessing language.

2.4.1 The Network Model. An example of the proposed schema definition language for describing a network database is given in Figure 4. The schema name is COMPANY. The NAME items and the LOC item have character string values; AGE values are integers having three significant digits, and SAL values are decimal fractions having eight significant digits with two decimal positions. The ORDER SORTED statements specify that member records are to be ordered by data item values rather than by some chronological order of insertion or application dependent positioning; the KEY clause in each member record description specifies the member data items that determine this order. In the PAYROLL set, the DUPLICATES phrase specifies that member records having identical

salaries are allowed as duplicates, and the relative ordering of duplicates is determined by system default. In the ALPHA set, the DUPLICATES phrase specifies that two member employees are not allowed to have the same name. The INSERTION and RETENTION clauses specify insertion and retention integrity constraints for member records of the set type. The differences among automatic, manual, or structural insertion and fixed, mandatory, or optional retention are discussed further in Section 4.1.6.

Defining the Example Database
In the Network Model

```
SCHEMA COMPANY
ENVIRONMENT PSEUDOLANGUAGE
RECORD DEPT
    NAME  CHARACTER 3
    LOC   CHARACTER 2
RECORD EMP
    NAME  CHARACTER 15
    AGE   FIXED 3 0
    SAL   FIXED 8 2
RECORD SUPERVISORS
    POSITION     CHARACTER 20
    STAFFSIZE   FIXED 3 0
    STARTDATE   FIXED 6 0
    ENDDATE     FIXED 6 0
RECORD STAFF
    POSITION     CHARACTER 20
    MANAGER      CHARACTER 15
    STARTDATE   FIXED 6 0
    ENDDATE     FIXED 6 0
SET PAYROLL
    OWNER DEPT
    ORDER SORTED
        DUPLICATES DEFAULT
    MEMBER EMP
        INSERTION MANUAL RETENTION MANDATORY
        KEY DESCENDING SAL
SET ALPHA
    OWNER SYSTEM
    ORDER SORTED
        DUPLICATES PROHIBITED
    MEMBER EMP
        INSERTION AUTOMATIC RETENTION FIXED
        KEY ASCENDING NAME
```

```
SET MANAGES
    OWNER EMP
    ORDER SORTED
        DUPLICATES PROHIBITED
    MEMBER EMP
        INSERTION MANUAL RETENTION OPTIONAL
        KEY ASCENDING NAME
SET JOBHISTORY
    OWNER EMP
    ORDER SORTED
        DUPLICATES PROHIBITED
    MEMBER SUPERVISORS
        INSERTION AUTOMATIC RETENTION FIXED
        KEY DESCENDING STARTDATE
    MEMBER STAFF
        INSERTION AUTOMATIC RETENTION FIXED
        KEY DESCENDING STARTDATE
```

Figure 4

2.4.2 The Relational Model. Unlike the network model, the relational model does not contain specific schema and subschema definition languages. Its major data definition statement is CREATE TABLE, which includes in its syntax the statement form for defining columns and constraints. Figure 5 shows the syntax for defining a particular version of the sample database. This syntax defines tables without loading any data. For each table there is a CREATE TABLE statement that names the table and names and defines its columns. In the definition of EMP, for example, NAME is a column whose values are character strings of length 15, AGE can assume integer values from 0 to 999, SAL takes on fixed decimal values from 0.00 to 999999.99, DEPT takes character strings of length 3, and MANAGER is like NAME. In addition, values in the NAME column of a given row must be unique and non-null. Finally, values in the DEPT column are tied by an integrity constraint to values in the NAME column of the DEPT table.

Defining the Example Database
In the Relational Model

```
CREATE TABLE DEPT
ENVIRONMENT PSEUDOLANGUAGE
  NAME  CHARACTER 3   NOT NULL   UNIQUE
  LOC   CHARACTER 2   NOT NULL   DEFAULT "NY"

CREATE TABLE EMP
ENVIRONMENT PSEUDOLANGUAGE
  NAME      CHARACTER 15  NOT NULL   UNIQUE
  AGE       FIXED 3 0
  SAL       FIXED 8 2
  DEPT      CHARACTER 3   REFERENCES DEPT.NAME
                          CASCADE MODIFY RESTRICT ERASE
  MANAGER   CHARACTER 15

CREATE TABLE SUPERVISORS
ENVIRONMENT PSEUDOLANGUAGE
  NAME         CHARACTER 15 NOT NULL REFERENCES EMP.NAME
                            CASCADE MODIFY CASCADE ERASE
  POSITION     CHARACTER 20 NOT NULL
  STAFFSIZE    FIXED 3 0
  STARTDATE    FIXED 6 0    NOT NULL
  ENDDATE      FIXED 6 0

CREATE TABLE STAFF
ENVIRONMENT PSEUDOLANGUAGE
  NAME         CHARACTER 15 NOT NULL REFERENCES EMP.NAME
                            CASCADE MODIFY CASCADE ERASE
  POSITION     CHARACTER 20 NOT NULL
  MANAGER      CHARACTER 15
  STARTDATE    FIXED 6 0    NOT NULL
  ENDDATE      FIXED 6 0
```

Figure 5

2.4.3 Other Models. Schema definition for other data models is usually similar in form to the examples given above for the network and relational models. Since there is currently no candidate standard for other models, the exact syntax will vary from one product to the next. Vendors of hierarchical systems must provide a way of defining the structure of various nodes and their relationships to each other. Similarly, the entity-relationship model must enable users to define entities, attributes, and relationships. Whatever the structure of the data model, the DBMS will have to have not only constructs that are similar to records and

components, but also a means of defining these structures and any relationships between them.

3. DATA MANIPULATION

In addition to data structures, a data model specifies operations that insert, delete, or modify data in a database. The specification includes both the primitive operations of the model and a mechanism for calling those operations from an external source. Whether that source is a batch program, an interactive query language invoked from a terminal, or an entirely separate application system is irrelevant to the model, although it may be very important to a potential user trying to discriminate between competing commercial DBMS's. The following sections discuss some of the possibilities for data manipulation and give explicit examples of operations on the network and relational databases defined in Chapter 2.

3.1 Modules and Procedures

As defined by ANSC X3H2, a module is a persistent object specified by either the Network Database Language (NDL) or the Relational Database Language (RDL) procedure language. It consists of a language environment clause and one or more procedure definitions, together with cursor declarations if it is a relational module, and subschema and keeplist declarations if it is a network module. The language environment clause specifies the name of a standard language (e.g., COBOL) from which procedures in the module will be called. A procedure definition consists of the procedure name, a sequence of parameter declarations, and a sequence of statements for querying or modifying the database. The declaration of a parameter specifies its data type and, in a network module, specifies whether it is an elementary value or an array of values. A parameter either assumes or supplies values to a corresponding argument in a call of the procedure.

Every module is associated with an application program. A procedure in the module is referenced by an external "call" from the application program. The call specifies the procedure name and supplies a sequence of parameter values corresponding in number and in data type to the parameter

declarations of that procedure. Each call of a procedure causes its sequence of statements to be executed. The parameters return values from the database to variables in the application program referenced by arguments in the call of the procedure.

Database modules and named procedures provide external programs direct access to NDL and RDL statements with no required additions or modifications to the syntax of the accessing language. All that is needed is a correspondence between data types and the ability to call separately compiled procedures written in a different language. Programming languages may use these facilities directly for interface to the DBMS, or they may call procedures implicitly either by preprocessing embedded database statements or by defining native syntax for invoking DBMS functions.

3.2 Database Access from External Languages

As noted above, programming languages or application systems accessing a database need the following:

* A correspondence between data types.

* A method for calling DBMS procedures.

A standard specification for these items is analogous to the problem of a standard specification for cross-language calls between any two programming languages. A type correspondence between DBMS data types and programming language data types specifies the language data types that can be passed validly as parameters to a DBMS procedure. This type correspondence for ANSI COBOL, FORTRAN, PL/1, and Pascal is specified by the NDL and RDL proposals. Type correspondence in general can range from the very strict, with a one-to-one mapping of data types, to the very flexible, involving significant run-time conversions. The examples in Section 3.3 contain a type declaration on each side of the interface, with an implicit assumption that the underlying type correspondence is well-specified.

The DBMS procedure calling requirement can be satisfied in several ways, including:

* Explicit procedures written and called directly by the user.

* Implicit procedures embedded in a calling program and extracted by a preprocessor.

* Native syntax defined as an extension to a programming language.

The following sections discuss each of these alternatives, and Section 3.3 provides simple examples.

3.2.1 Explicit Procedure Approach. The "explicit procedure" approach is available to any language having a subroutine facility capable of calling separately compiled procedures written in a different language. In this case the end user writes both an application program and a separate DBMS module for each database application. A standard programming language compiler compiles the application program, and the DBMS itself compiles the DBMS module. The module contains named database procedures, each consisting of a sequence of DBMS functions written in the database procedure language syntax. Using its standard syntax for invoking subroutines, the application program calls a database procedure by name. Variables defined in the application program and passed to a DBMS procedure are declared as parameters in that procedure. A method for linking the compiled language program with the compiled DBMS module is implementor-defined.

3.2.2 Implicit Procedure Approach.

The "implicit procedure" approach assumes specification of some variation of a preprocessor. References to DBMS functions are embedded directly in the application program using some convention to distinguish DBMS statements from standard programming language statements. In this case, the programmer writes a single program containing a mixture of programming language statements and database language statements. The program is processed by a precompiler to produce a "pure" external program capable of compilation by a standard programming language compiler, and a database module, as above, capable of compilation by the DBMS. A method of generating names for the DBMS procedures and calling them by that name from the "pure" program is included in the specification of the preprocessor; the original source application, written by the end user, need not provide procedure names.

3.2.3 Native Syntax Approach. The "native syntax" approach involves adding specific database syntax to existing programming languages. This syntax could be any of the following:

* Native syntax for each DBMS function.

* New syntax for combinations of DBMS functions.

* Syntactic variations for triggering DBMS calls.

In any case, the programmer writes a single program containing integrated syntax for programming language statements and database functions. The complete program is compiled by a standard programming language compiler to produce object code for a database session. If a language defines new combination functions, then each such function must be definable by a sequence of DBMS functions. A programming language developer would have the option of defining syntactic variations for calling DBMS functions. For example, an NDL test function could be triggered by a native language Boolean expression with the test result used in program control statements. A programming language could also require a one-to-one correspondence between selected programming language variables and database data items. Such a correspondence would minimize the native language syntax needed for parameter passing between the DBMS and the accessing language. If desired, the programming language could include a facility for handling automatically all exception conditions so that the application programmer would not have to check manually the database status after each call to a DBMS function.

3.3 Examples

The rest of this chapter consists of a number of examples designed to illustrate some of the more common operations of the network and relational models. Other database models will probably be analogous in some ways to one of these two models. Navigational systems will have operations that resemble those of the network model, and systems that rely on retrieval through indexes will probably provide a selection language somewhat similar to the relational calculus. Whatever the model, the interface it provides to host programming languages may fall into one of the categories discussed above.

3.3.1 Explicit Procedure Calls. Each explicit procedure example consists of a pure programming language main program with calls to DBMS procedures contained in a separate DBMS module. The main program is written entirely in the standard syntax of a programming language; all variables are defined with programming language data types, and all subroutine parameters are declared and passed to the external procedures in the manner standard to that programming language. The database procedures are written in the proposed database language and are combined together into a single database module that can be processed separately by the DBMS. Each database procedure has a name that is referenced in the call from the main program. Procedures declare data types for each parameter passed. Each procedure consists of a sequence of DBMS statements that use the declared parameters to invoke DBMS functions.

Figure 6 is an example application for loading a network database defined by the COMPANY schema through the following PERSONNEL subschema:

```
SUBSCHEMA PERSONNEL IN COMPANY
   RECORD DEPT ALL
   RECORD EMP ALL
   SET PAYROLL
   SET ALPHA
```

The program initiates a database session, inputs values for DEPT and EMP data items from an external device, and then creates and stores new department and employee records in the database. The main program consists of four calls to database procedures. In line 8, READYDEPTEMP calls DBMS functions that ready the DEPT and EMP record types for protected update. In lines 11 and 14, STOREDEPT and STOREEMP pass parameters that carry values for data items of department and employee records. The corresponding database procedures then invoke DBMS functions to store the records in the database and to connect the records to any sets that require manual connection. In line 17, FINISHLOAD calls DBMS functions that commit the modifications of the session to the database and finish access to the previously opened record types.

For the relational example we assume that the relational database portrayed in Figure 2 has been loaded in a manner similar to that used to load the network database. Figure 7 is then a relational application that uses explicit procedures to select company employees whose current annual salary in thousands is less than twice their current age, to print the name of each such employee, and then to modify the

Sample Program to Load a Network Database
Using Explicitly Declared Procedures

```
1.    Declare
2.              DEPTNAME     PIC X(3)
3.              DEPTLOC      PIC X(2)
4.              EMPNAME      PIC X(15)
5.              EMPAGE       PIC 999
6.              EMPSAL       PIC 9(6)V99  ;
7.    Begin Main Program
8.    Call READYDEPTEMP  ;
9.    While (Not EOF )Do
10.       Accept  DEPTNAME,DEPTLOC ;
11.       Call STOREDEPT Using DEPTNAME DEPTLOC ;
12.       While (Not EOF  ) Do
13.           Accept EMPNAME, EMPAGE, EMPSAL ;
14.           Call STOREEMP Using EMPNAME EMPAGE EMPSAL ;
15.       Endwhile
16.   Endwhile;
17.   Call FINISHLOAD
18.   End Main Program.
```

```
1.      MODULE
2.        ENVIRONMENT PSEUDO-LANGUAGE
3.        SUBSCHEMA PERSONNEL IN COMPANY
4.      PROCEDURE READYDEPTEMP
5.        READY DEPT SHARED UPDATE
6.        READY EMP SHARED UPDATE
7.      PROCEDURE STOREDEPT
8.            N  CHARACTER 3
9.            L  CHARACTER 2
10.       STORE DEPT
11.           SET NAME TO N
12.           SET LOC TO L
13.     PROCEDURE STOREEMP
14.           N  CHARACTER 15
15.           A  FIXED 3 0
16.           S  FIXED 8 2
17.       STORE EMP
18.            SET NAME TO N
19.            SET AGE TO A
20.            SET SAL TO S
21.       CONNECT EMP TO PAYROLL
22.     PROCEDURE FINISHLOAD
23.       COMMIT
24.       FINISH ALL
```

Figure 6

Sample Program to Modify a Relational Database
Using Explicitly Declared Procedures

```
1.    Declare EMPNAME   PIC X(15)
2.            EMPAGE    PIC 999
3.            EMPSAL    PIC 9(6)V99
4.            RDLCODE   PIC 9(5) ;
5.    Begin Main Program
6.    Call EMPOPEN Using RDLCODE ;
7.    If (RDLCODE <> "00000") Then
8.        Begin Print "DB-ERROR" ; STOP End ;
9.    Call EMPFETCH
10.       Using EMPNAME  EMPAGE   EMPSAL   RDLCODE ;
11.   While (RDLCODE = "00000") Do
12.       Print EMPNAME ;
13.       Call EMPMODIFY Using EMPSAL RDLCODE ;
14.       Call EMPFETCH
15.            Using EMPNAME EMPAGE EMPSAL RDLCODE ;
16.   Endwhile;
17.   If (RDLCODE = "00100") Then
18.       Begin Call EMPROLLBACK ;
19.       Print "DB-ERROR"; STOP End ;
20.   Call EMPCOMMIT ;
21.   End Main Program.
      1.   MODULE
      2.   ENVIRONMENT PSEUDOLANGUAGE
      3.   PROCEDURE EMPOPEN
      4.        STATUS
      5.      OPEN RAISEPAY CURSOR FOR
      6.           SELECT NAME,AGE,SAL
      7.                FROM EMP
      8.                WHERE  SAL < 2*AGE*1000
      9.                FOR MODIFY OF SAL
      10.   PROCEDURE EMPFETCH
      11.       N CHARACTER(15)
      12.       A FIXED (3,10)
      13.       S FIXED (8,2)
      14.       STATUS
      15.     FETCH RAISEPAY INTO N,A,S
      16.   PROCEDURE EMPMODIFY
      17.       S FIXED (8,2)
      18.       STATUS
      19.     MODIFY EMP
      20.       SET SAL = S + 5000
      21.       WHERE EMP IS CURRENT OF RAISEPAY
      22.   PROCEDURE EMPCOMMIT
      23.     COMMIT
      24.   PROCEDURE EMPROLLBACK
      25.     ROLLBACK
```

Figure 7

The chief advantage of the implicit procedure approach is the convenience to the end programmer of being able to write self-contained application programs. Specifications for the preprocessor would define how NDL or RDL-MODULE statements and NDL or RDL-CORRESPONDENCE statements produce the resulting DBMS module with proper parameters for DBMS procedures. The chief disadvantage of this approach is that at present there is no standard specification for forming a database module from an application program containing embedded database statements; procedures could be defined in a number of different ways resulting in potentially different error messages. Another drawback is that the application program variables are logically separate from the database parameters, thus requiring explicit SET clauses in database statements involving parameter passing between the DBMS and the accessing language. In the absence of a standard specification for a preprocessor, the explicit procedure approach given above circumvents the chief disadvantage. In some cases, an integrated approach where DBMS function syntax is combined with programming language syntax, and database data items are uniquely identified with programming language variables, may be desirable. An example of such an approach is given in the next section.

3.5 Native Syntax

This approach for programming language interface to a database requires specifications for invoking DBMS functions directly as part of the programming language. Syntax for calling these functions is integrated into existing syntax of the language, thus requiring modifications to the programming language compiler for handling new verbs, new conditions, and possibly new reserved words. The programming language designers may choose to integrate database exception conditions into the normal exception handling capabilities of the language, or to incorporate database test statements into the normal control statements of the language.

Figure 10 is an example program over our network database for achieving the same employee salary modification as in the preceding examples. It is a COBOL program with integrated DBMS functions written in proposed COBOL DML syntax [X3J483]. The SUB-SCHEMA SECTION of each program names the subschema to be processed, names the record types and set types used as COBOL DML parameters in the program, names any keeplists, and provides a one-to-one correspondence between subschema records and data items and COBOL records and data items. The effect of the data item correspondence is that each execution of a COBOL DML GET, MODIFY, or STORE

Sample Program to Modify a Network Database
Using COBOL and COBOL DML Syntax

```
1.    IDENTIFICATION DIVISION.
2.    PROGRAM-ID.  MODSAL.
3.    ENVIRONMENT DIVISION.
4.    DATA DIVISION.
5.    SUB-SCHEMA SECTION.
6.    DB  PERSONNEL WITHIN COMPANY.
7.    LD  TEMP.
8.    RD  RECORD SECTION.
9.        01  EMP.
10.            02 NAME PIC X(15).
11.            02 AGE  PIC 999.
12.            02 SAL  PIC 9(6)V99.
13.    SD  ALPHA.
14.    WORKING-STORAGE SECTION.
15.    77  LASTREC  PIC X(3)   VALUE 'NO'.
16.    PROCEDURE DIVISION.
17.    DECLARATIVES.
18.    LASTREC-ERROR SECTION.
19.        USE FOR DB-EXCEPTION ON '00100'.
20.        MOVE 'YES' TO LASTREC.
21.    ABNORMAL-ERROR SECTION.
22.        USE FOR DB-EXCEPTION ON OTHER.
23.        DISPLAY 'DB ERROR'.  STOP RUN.
24.    END DECLARATIVES.
25.    MODIFY-DATABASE.
26.        READY EMP USAGE-MODE IS PROTECTED RETRIEVAL.
27.        IF ALPHA IS EMPTY STOP RUN.
28.        FIND FIRST EMP WITHIN ALPHA.
29.        PERFORM GET-REC UNTIL LASTREC = 'YES'.
30.        END-PERFORM.
31.        READY EMP USAGE-MODE IS PROTECTED UPDATE.
32.        FIND FOR UPDATE FIRST WITHIN TEMP.
33.        MOVE "NO" TO LASTREC.
34.        PERFORM MODIFY-REC UNTIL LASTREC = "YES".
35.        COMMIT.  FINISH EMP.  STOP RUN.
36.    GET-REC.
37.        GET EMP.
38.        IF SAL < 2*AGE*1000; KEEP CURRENT USING TEMP.
39.        FIND NEXT EMP WITHIN ALPHA.
40.    MODIFY-REC.
41.        GET EMP.
42.        ADD 5000 TO SAL.  DISPLAY NAME OF EMP.
43.        MODIFY EMP.
44.        FREE FIRST WITHIN TEMP.
45.        FIND FOR UPDATE FIRST WITHIN TEMP.
46.    END-PERFORM.
```

Figure 10

statement is equivalent to the corresponding NDL Statement with implicit SET clauses for transferring data values between a database record and a COBOL record.

The type declaration for each data item in the SUB-SCHEMA SECTION must match the type declared for that item in the schema. Lines 17-24 of Figure 10, DECLARATIVES, provide for automatic processing of database exception conditions so that the programmer need not check the status register after each invocation of a DBMS function. According to COBOL DML rules every possible exception condition must correspond to exactly one USE FOR statement.

4. FEATURES OF THE NETWORK AND RELATIONAL MODELS

Chapters 2 and 3 covered the basic structures and operations of the network and relational models, but both of them have other features that seem more complex or require further discussion. This chapter focuses on such characteristics as arrays, recursive sets, cursors, nested queries, and triggers. It also considers some of the advantages and disadvantages of each model -- i.e., what it does well and what it does only adequately or even poorly.

4.1 The Network Model

In its original CODASYL form the network model was designed for handling data that batch COBOL programs could process one record at a time. The current ANSI specification has diverged in a number of ways from its ancestor, but the basic orientation toward sequential processing still remains. Cursors, keeplists, and session states enable the DBMS to keep track of its position as it navigates through the database. Though the bias toward COBOL is gone from the syntax, many of the same operations are available, and with a few exceptions the data structures are the same as always.

4.1.1 Cursors and Session State. The logical structures of a network database may be represented graphically as a collection of tables showing the record occurrences associated with each record type and the set occurrences associated with each set type. A table representation of a database whose logical structures are defined by the PERSONNEL

subschema is given in Figure 11. Each record is identified by a unique identifier called a database key. The database key is never directly accessible by an end user but is used to define the effect of DBMS functions on the session state (see Figure 12) during a database session. A set is represented by the database keys of its owner and member records. In the set type tables of Figure 11, each owner database key is associated with a sequence of member database keys. The order of the sequence represents the order of member records within the set.

The action initiated by a DBMS function is dependent upon values of cursors, keeplists, and ready lists contained in an associated session state. Figure 12 is a representation of the session state specified by the Interpretive State section of the current draft proposed NDL for a database module acting over the PERSONNEL subschema. The session state is prepared by the DBMS prior to execution of the first procedure in a database session, and updated whenever DBMS statements change the state.

The cursors identify a single record from each record type and a single set from each set type of the subschema. Each record type cursor contains the database key of the current record of that record type; each set type cursor contains the database key of the owner record of the current set of that set type together with the database key of its current member; and the session cursor contains the database key of the current record of the session. A keeplist, which is a sequence of database keys that can be used by a programmer to save references to individual records, is maintained for each keeplist named in the associated module. The ready list, which is maintained by the DBMS to help in managing shared access to the database, is a list of record names together with their share specifications.

4.1.2 Keeplists. As stated above, a keeplist is a sequence of database keys. Some database statements add database keys to the end of a named keeplist, and the user program has access to both the front and the end of the sequence. Keeplists, which are declared in a module, enable an application program to retain and use database keys for quick retrieval of records to be examined or modified. They therefore prevent the program from having to repeat searches for records that will be used again. In Figure 10, lines 36-39, the procedure GET-REC tests the current EMP record and, if it qualifies, adds its database key to the keeplist TEMP for further work. After the last call to GET-REC, the procedure MODIFY-DATABASE finds the first key in TEMP (line 32) and then calls the procedure MODIFY-REC, which modifies the

NETWORK DATABASE OCCURRENCES

IN THE PERSONNEL SUBSCHEMA

EMP Records

DB-KEY	NAME	AGE	SAL
1	Joe	21	18K
2	Adam	36	39K
3	Jane	18	38K
4	Sally	32	65K

DEPT Records

DB-KEY	NAME	LOC
5	MKT	NY
6	MFG	LA

PAYROLL Sets

OWNER	MEMBERS
5	2, 3
6	4, 1

MANAGES Sets

OWNER	MEMBERS
2	3
4	1

ALPHA Set

OWNER	MEMBERS
sys	2,3,1,4

Figure 11

Session State

For the Network Database

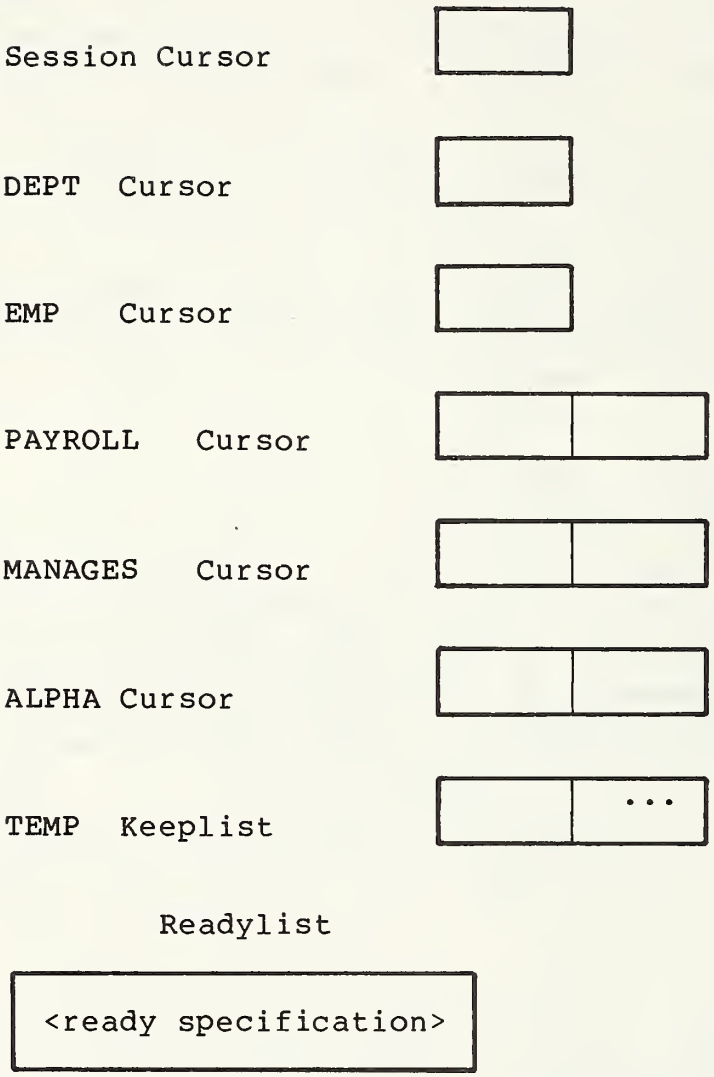


Figure 12

record, frees the database key from the keeplist (line 44), finds the next key in TEMP, and then repeats the process for the new record.

4.1.3 Arrays. As noted in Section 2.2.5, arrays have replaced repeating groups in the ANSI specification of the network model. None of the examples shows the function of arrays, but it is not difficult to envision a situation in which they would be useful. Although they are less flexible and powerful than sets, they offer some advantages in simplicity for data of fixed dimensions. For example, a record type containing information about computer users might have a component named TERMCHARS that would consist of ASCII characters whose sequence in the array would define the user's terminal for certain application programs. Keeping this information in such a record would avoid the necessity of traversing a set and locating a separate record with the same information. In a scientific database the importance of arrays would be even greater.

4.1.4 Recursive Sets. Recursive sets are simply sets in which the owner record type is identical to one of the member record types. In Figure 1 MANAGES is a recursive set type. Each of its occurrences includes an EMP record as owner and the records of all employees who work directly under the person identified by the owner record. The word "directly" is critical here, since MANAGES sets obey the same uniqueness rules as any other set. No EMP record can be a member of more than one MANAGES set. Recursive sets make it easy to answer a number of questions about the relationships between records.

For example, to get a list of every employee whose immediate manager is under 30, we use the ALPHA set to examine each employee record. If the AGE component is less than 30, we add to our result the member records of the MANAGES set associated with the current record. When we finish traversing the ALPHA set, we have our answer. To get a similar list of all employees who are older than their immediate managers, we traverse ALPHA again, but this time we have to traverse every MANAGES set to select those member records whose AGE component is smaller than the AGE component of the owner record. An alternative form would require traversing ALPHA and for each employee record doing a find owner using MANAGES to obtain the record for the comparison of ages. If the task were to get a list of all employees who have a younger manager at any level, we would again traverse ALPHA one record at a time. This time, however, we would find the owner of the MANAGES set recursively until either the

starting EMP record qualified or we reached the top of the organizational structure. In all of these cases we would be taking advantage of existing data structures to navigate efficiently through the database.

4.1.5 Singular Sets. Singular sets -- sets whose types are defined with SYSTEM as the owner record type -- provide ordered access to member record types. Since there is only one record occurrence of the type SYSTEM, there is one set (hence, "singular") that contains all member records of each named type. Singular sets thus provide the same functionality as record order keys (one member record type) and areas (multiple member record types) in the CODASYL model. If an organization needs a record type sorted in several ways, then the database administrator simply defines a new singular set for each ordering. In the example database shown in Figure 1, ALPHA orders EMP records alphabetically according to the NAME component and thus supplements the ordering by department and salary defined for PAYROLL sets. Host-language programs can use whichever order is most appropriate for the application.

By default, the network specifications provide direct access to each record type that is essentially equivalent to the existence of a singular set type with that record type as its only member and having system default order. Thus, in Figure 1, it would be possible to find SUPERVISORS descriptions that satisfy a certain condition by navigating directly through the record type without first navigating through DEPT and EMP record types.

4.1.6 Set Insertion and Retention. Set insertion governs the initial insertion of member records into sets and may be AUTOMATIC, MANUAL, or STRUCTURAL. If insertion is AUTOMATIC, then whenever a new occurrence of the member record type is stored in the database, it is automatically connected to the set identified in the session state by the set cursor corresponding to the set type in which the member record type is defined. This is the option chosen for SUPERVISORS and STAFF records in the network schema of Figure 4. As each new STAFF or SUPERVISORS record is stored in the database, it is connected automatically to the JOBHISTORY set identified by that set cursor in the session state. A set cursor may be positioned to the set owned by an EMP record simply by finding that record. If insertion is MANUAL, then records may be inserted as member records in a set only by an explicit connect statement. This is the insertion option specified for EMP records as members of both PAYROLL and MANAGES sets in the example. Finally, for STRUCTURAL

insertion, the effect on a new member record is similar to AUTOMATIC insertion, with the exception that the receiving set is not identified by a set cursor. Instead it is selected so that its owner record has values for certain data components equal to corresponding values in the new member record. For example, if the EMP record type had a DEPT component as in the relational example, then it could be defined as a structural member of the PAYROLL set type with the structural specification defined as DEPT of EMP equal to NAME of DEPT. If such were the case, then line 21 of Figure 6 would not be necessary as the connection would take place automatically.

STRUCTURAL insertion also has an effect on the modify statement; if a structural data item of a member record type is modified, then it automatically becomes a member of a new set of the same set type having an owner record that satisfies the structural condition. In the above modification of the network example this would mean that any modification of the DEPT component in an EMP record would trigger a reconnection of that employee to a DEPT record with a matching department NAME. In all of the above situations, the sequential position of the new member record in its set is determined by the ordering criteria defined for that set type.

Set retention governs the disposition of member records after they have been inserted into some set. Retention may be FIXED, MANDATORY, or OPTIONAL. For FIXED retention, a member record remains a member of the set of first insertion for its entire lifetime in the database. This is the option chosen for SUPERVISORS and STAFF records in the network schema of Figure 4. This declaration has the effect of linking job history descriptions permanently to a specific employee; if an employee record is erased from the database, then all job history information pertaining to that employee is erased at the same time.

Whenever MANDATORY retention is specified, a record, having once become a member record of a set, remains a member of some set of the same set type until it is erased from the database. This is the retention option specified for EMP records as members of PAYROLL sets in the network example. The effect of this declaration is that once an employee record is first assigned to some department it must remain forever assigned to that department or to some other department in the database; it can never become unattached. Of course, since insertion is MANUAL, a new employee record remains unattached to any department until it is first inserted by a specific connect statement. Because of MANDATORY retention for employees, no department record may be

erased from the database without first erasing, or connecting to other departments, all employee records in the PAYROLL set owned by that department. If desired, such erasure could be effected with a single erase statement using the full cascade option. In that situation all employees connected to the erased department would be erased, and the effect of that erasure would cascade to the job history and all other records subordinate to the erased department record.

With OPTIONAL retention, a member record may remain a member of the set it was connected to, or it may be removed either for reconnection to some other set or to become unassociated with any owner record. This option is specified for the MANAGES set type in the example. Thus an employee may or may not have a manager recorded in the database, depending on whether or not the EMP record is connected via some MANAGES set to another employee. Under OPTIONAL retention, if an EMP record is erased, then the effect may or may not cascade to subordinate employees, depending on whether the erase statement specifies full or partial cascade.

4.1.7 Set Ordering. Set ordering is declared in the schema for each set type; it specifies a sequential ordering for the member records of each set. The ordering may be SORTED by component values, in which case additional KEY clauses specify the sort key and direction, or it may be one of the following: chronological order as FIRST or LAST when compared with previous insertions; relative positioning as NEXT or PRIOR to a given member record; or system DEFAULT order determined by the implementation. The schema in Figure 4 shows only the SORTED option for each member record type, but if JOBHISTORY member records were inserted chronologically with respect to former job positions, then the effect of order FIRST (i.e. most recent insertion put first) would be equivalent. Likewise, in the ALPHA set for employee records, if each new employee record were inserted in its correct alphabetical position, then order NEXT or order PRIOR would be equivalent to order SORTED by ascending NAME. System DEFAULT order, which allows the DBMS to optimize retrievals without having to provide a specified order, will be the option of choice when no particular order is needed by users. The only requirement for the DBMS in DEFAULT order is that it be reproducible within a single transaction; between transactions the implementation may reorder the member records of any such set.

Even in multiple member set types, like JOBHISTORY in the example, there is only a single sequential order for the member records of each set. In that example, the SUPERVISORS and STAFF member records might be intermixed since STARTDATE is a common key item. If the database administrator wished to partition job history descriptions by job category -- supervisor or staff -- then those member record types would be ordered by an optional declaration in the SORTED clause.

4.1.8 Complex Data Structures. As the previous sections have demonstrated, the network model has a variety of complex and powerful data structures. While the use of pointers like database keys enables the DBMS to navigate through a database efficiently, the complexity of sets, cursors, keeplists, and other structures makes the network model better suited to production applications than to ad hoc data manipulation. Many people who have little or no programming experience could find the data structures of the network model difficult to learn and use. Such complexity can also make it expensive to redefine data structures. With so much information already encoded, the network model probably works best for "static" databases, where structures and programs that navigate through them have a relatively long life span. The values of the database could change frequently, but the structures should probably be stable.

4.2 The Relational Model

From its beginning the relational model has developed independently of any programming language. Its theoreticians and developers have consistently focused on the needs of interactive users who might prefer not to write programs in host language. This focus remains in the current ANSI draft, though of course the specification does provide for interfaces with traditional programming languages. The relational model has repeatedly emphasized two features: the simplicity of presenting data in tabular form, and the mathematical soundness of table definition and manipulation. Each of the following sections discusses either a feature of the model or some consequences of its mathematical elegance.

4.2.1 Nesting and Range Variables. Relational DBMS's offer two complementary techniques for answering recursive queries that the network model would handle with recursive sets. The first of these involves variables that range over specified tables and enable the user to make comparisons such as

X.NAME = Y.MANAGER. In the current draft of the ANSI relational specification these variables can be implicit -- the table names themselves -- or explicitly declared synonyms with a limited scope. A second technique involves nested queries, in which the WHERE clause of one query includes a sub-query. The result of the sub-query must be a table with exactly one column.

For example, a user of the sample database could find all the employees whose immediate managers are younger than 30 by running the query in Figure 13(a). Though the implementation of this query could vary, the logic would remain the same. The sub-query returns a one-column table listing the names of all employees under 30. The main query then examines every row of EMP and selects the employee name and manager whenever the manager is in the table of names returned by the sub-query. Logically, a relational DBMS would have to retrieve and examine every row in EMP twice, once for the sub-query and once for the query. For n rows in EMP, the execution time would be proportional to 2n. Of course, an index on AGE could substantially reduce the execution time of the sub-query, but execution time for the whole query would still be O(n).

In the previous example the predicate within the sub-query was absolute -- i.e., it involved a column value and a constant. When the predicate becomes relative, as in Figure 13(b), the query is more complicated. In this case we want to know the name and manager of every employee who is older than his manager. Logically, the DBMS must retrieve each row of EMP and then search EMP again for the row whose NAME column matches the MANAGER column of the retrieved row. If the result of the sub-query is more than a single value, it is an error. For n rows in EMP, the execution time is proportional to n**2, and an index on AGE is unlikely to improve the speed.

Both of the previous queries involved a single nested sub-query. The relational specification allows nesting to any level, but that level must be fixed in a given query. In the current draft of the standard, there is no way to formulate a query that will return the names of all employees who are older than at least one of their managers, immediate or higher. Because the specification omits control structures, a user who needs the answer to this question must write a program that climbs the organizational tree implied by the MANAGER column in the EMP table. A query in the relational data manipulation language might require two levels of nesting for some employees and five levels for others. A host language program, however, can retrieve rows one at a time and store in its own data structures the

```
SELECT   NAME, MANAGER
FROM     EMP
WHERE    EMP.MANAGER IN

               (SELECT   NAME
                FROM     EMP
                WHERE    EMP.AGE < 30)
```

(a)

```
SELECT   NAME, MANAGER
FROM     EMP AS X
WHERE    X.AGE >

               (SELECT   AGE
                FROM     EMP AS Y
                WHERE    X.MANAGER = Y.NAME)
```

(b)

Figure 13

necessary data to answer the query.

4.2.2 Views. A view in the relational model is analogous to a subschema in the network model -- i.e., it defines a portion of the database as it will appear to particular users. Each view is a derived table because it results from the execution of a statement, namely a SELECT statement; it differs from other derived tables because the SELECT statement (actually, the <query spec>) that defines it is a permanent part of the database rather than just an operation during a user session. If the <query spec> selects columns from only a single table without a sub-query, then the resulting view is updatable; otherwise, it is read-only. A database administrator may wish to use the CREATE VIEW statement to establish particular views of the data for users with similar privileges or duties.

For example, Figure 14 shows the syntax for defining a view of the example relational database that would be suitable for employees who need information about the current position of both supervisors and staff. Here the <query spec> is the union of two SELECT statements, each of which joins the EMP table with one of the two tables containing the job history of current employees. For each employee only one row from either SUPERVISORS or STAFF will satisfy the condition that ENDDATE be null. The view definition will join the POSITION and STARTDATE columns from that row with the row from the EMP table that has the same value in the NAME column, and the result will be a derived table with the columns shown in the figure. Because the view results from more than a single table, it is not directly updatable. It could, however, be used as a source for comparisons with one of the base tables, which could then be updated by users with appropriate privileges.

4.2.3 Triggers. Because relational databases depend so heavily on operations rather than structures to maintain relationships among data values, they invariably have a high degree of data redundancy. Frequent modification of data values that occur in several tables could result in an inconsistent database. To avoid this problem, the current ANSI specification provides referential integrity, which is a technique for tying together corresponding data from different tables. In the example database defined in Figure 5, for example, modification of an employee's name or deletion of an employee's record from the EMP table would cascade through to the SUPERVISORS and STAFF tables. Any rows in those tables that refer to modified rows in the EMP table automatically undergo the same modification on the

Creating a View in the Relational Data Model

```
CREATE  VIEW    EMPJOBINFO

  AS    SELECT  EMP.ALL, SUPERVISORS.POSITION,
                   SUPERVISORS.STARTDATE

        FROM            EMP, SUPERVISORS

        WHERE           SUPERVISORS.NAME = EMP.NAME
                          AND SUPERVISORS.ENDDATE = NULL

    UNION

        SELECT  EMP.ALL, STAFF.POSITION, STAFF.STARTDATE

        FROM            EMP, STAFF

        WHERE           STAFF.NAME = EMP.NAME
                          AND STAFF.ENDDATE = NULL
```

EMPJOBINFO						
NAME	AGE	SAL	DEPT	MANAGER	POSITION	STARTDATE

Figure 14

corresponding columns. Any rows referring to deleted EMP records are also deleted.

The EMP table itself is in turn tied by a referential constraint to the DEPT table. Modifications to the NAME column in the DEPT table cascade through to the DEPT column in the EMP table. Deletions, however, work differently. The RESTRICT ERASE constraint on the DEPT column in the EMP table prohibits users from deleting department rows without first modifying or deleting any corresponding rows in the EMP table. In this case the constraint actually works in the opposite direction, preventing a modification that would result in an inconsistent database.

4.2.4 Flexibility. By relying on operations rather than structures for much of its semantics, the relational model makes possible a flexibility that neither the network model nor any other structure-oriented model can match. Users who have either a "dynamic" database or frequent ad-hoc requests for specialized information may find this flexibility very appealing. The current ANSI specification makes modifying and retrieving data very simple in a large number of cases, yet still enables a user to pose elaborate, complex queries without incurring the overhead of a complex host language program. In many environments such flexibility may match exactly the needs of the user. However, the reliance on retrieval by value rather than by logical pointers will probably keep the relational model from performing as well as the network model on "static" databases with infrequent structural changes but a lot of data processing.

4.2.5 Mathematical Precision and Elegance. Since Codd's original formulation the people who have done research on the relational model have in general preserved its firm mathematical basis. The builders of experimental and commercial systems have followed suit, though they have usually found it convenient or expedient to give up the elegant but sometimes impractical view of a table as a set of rows with no duplicates allowed. In its current draft the relational standard acknowledges the practical importance of allowing duplicate rows. It does insist, however, on another important feature of the model, the closure of tables under all operations on them. This feature ensures that the result of any relational operation is always another relation, or table. No matter how simple or complex a SELECT statement is, its result will still be a table and hence will be conceptually simple.

For many users the relational data language will seem straightforward for most operations. Its English-like version of the predicate calculus hides some mathematical constructs that can confuse a user who is unfamiliar with the language of logic and mathematics. The full power of the predicate calculus, however, still shows up in the WHERE clause, which specifies criteria for qualifying or selecting rows from appropriate tables. Because this clause is powerful and elegant enough to meet the needs of sophisticated users, it may sometimes prove difficult for less experienced people to learn and use. Database administrators may wish to construct menu-driven query capabilities for the most common requests by inexperienced users.

4.2.6 Performance. In its early experimental implementations the relational model gained a reputation for poor performance. Queries could take longer for the DBMS to answer than a user would need to search through printed output. Commercial implementations have improved performance considerably, often by adding indexes on columns that are frequently used in retrievals or by clustering values to optimize relational joins. Indexes and clustering may add a lot of overhead to relational DBMS's, but they can reduce response times to acceptable levels. In essence, an index on a column provides the relational model with pointers that are similar in function to those of the network model. Without these pointers the relational model must do a lot of searching to answer even some simple queries, and physical storage decisions become important factors in performance. Clustering often has the same physical characteristics as the implementation of network model sets. Because indexes and clustering affect only the performance of a DBMS and not its functionality, they are omitted from the standard and left to individual implementors.

5. SELECTION ISSUES

Choosing the right data model for a particular set of application requirements is almost certainly the first and most important step in the selection process. But it is not the last. This chapter discusses some of the other aspects of a DBMS that the buyer should also consider before actually purchasing a system. Different vendors may vary enormously in the markets they try to reach and in the features they offer their customers. Those features include user interfaces, support software, compatibility with hardware and

operating systems, and customer services. Only the potential buyer can decide which commercial product best meets the needs of the users' applications.

5.1 System Parameters

While the ANSI specifications of the network and relational models describe in detail available data structures and operations, they do not establish or even recommend values for a number of critical system parameters. A user of a relational system may want to add the requirement that the DBMS support at least a given number of columns per table, rows per table, tables per database, or even databases per operating DBMS. Similarly, a network user may need a minimum number of components per record type, records per record type, record types per database, sets per set type, or set types per database. Identifying appropriate values for these parameters could be the difference between meeting application requirements and buying the wrong product for the job. Understating the requirements could result in a system that will quickly be outgrown, and overstating them could result in an unnecessarily slow or awkward system.

There are also a number of parameters that are independent of data model. One application may require either very long or variable length character strings, while another may require either a given level of precision in real numbers or a large maximum for integer values. Some applications may require specialized data types like bit strings to represent video pictures or enumeration types to represent finite sets of alternatives. In some environments it may be critical for the DBMS to allow at least a given number of simultaneous users, either of the DBMS or of a particular database.

5.2 Hardware and Operating System Support

For many organizations one of the primary factors in selecting a DBMS may be compatibility with particular hardware and a specific operating system. In that case the purchaser should make sure that prospective products meet all of the organization's requirements in the existing hardware and software environments. Even the same commercial product may vary in important ways from one implementation to another. If the organization is procuring a computer, an operating system, and a DBMS simultaneously, or if it may soon change hardware or operating systems, then the

portability of prospective DBMS's is an issue. A product that runs on a variety of machines may give the customer important flexibility (e.g., in converting to another DBMS or to a distributed database) that could outweigh the immediate performance or usability of the DBMS.

Besides the possible requirement of compatibility with particular brands of hardware and software, users have to consider such features as the memory requirements of the DBMS. A microcomputer with only 64K of memory may not be suitable for a DBMS that requires 48K just for its executable image. Similar problems could occur on larger machines, where the DBMS might take up enough main memory to hamper its own operation or the operation of other programs running under the same operating system on the same processor. Since many DBMS's work with large databases, a prospective buyer may need to find a match between existing disk capacity and the secondary storage requirements of a particular product. DBMS's may vary considerably in the space overhead necessary for efficient performance. A customer needs to know whether a prospective DBMS can share a disk with other files, and whether the DBMS and operating system are compatible in their handling of files on disk. If efficient operation of the DBMS requires that files be laid out in contiguous sectors on disk, then either the operating system must allocate disk space that way or the DBMS must handle its own I/O.

5.3 Backup and Recovery Facilities

Establishing and operating a database are expensive and time-consuming projects, and the DBMS must provide adequate means to protect the user's investment. No matter how good the DBMS, the support software, and the computer system are, they will suffer occasional system crashes. Every DBMS should have some strategy for protecting the database against these failures. At the very least the user should demand a convenient way to backup the entire database in case of a disastrous loss of on-line data. Weekly full backups supplemented by daily incremental backups are one example of a backup strategy, but losing even a day's transactions on a busy database could be costly. Some products provide a journaling facility that keeps a record of all transactions since the last backup. The journal file may even be maintained on a separate disk to avoid its being scrambled at the same time as the database. In the case of minor crashes the DBMS can simply repeat all the transactions logged since the last successful update of the database on disk. For individual users some DBMS's, including

the proposed standards, provide commit points that enable a user to specify a series of database executions as a single transaction that is either completed successfully or not done at all. Further guidance on protecting data files is available in other publications available from the Institute for Computer Sciences and Technology (ICST) within the National Bureau of Standards (NBS) [FIPS79, FIPS81a].

5.4 Bulk Loading and Unloading

While a number of databases may have to be keyed in at a terminal, the user will often want to prepare data files without invoking the DBMS or to convert existing data files into a format suitable for it. Bulk loading of a DBMS from data files is probably a requirement for most production applications. A flexible loading utility that can handle a variety of input file formats could certainly enhance the attractiveness of a particular product. In some cases upgrades of the DBMS or even minor changes in the schema definition may require dumping the entire database and reloading it after the change. Under those circumstances, of course, bulk loading and dumping utilities would be a necessity.

5.5 Schema Manipulation

A number of DBMS's offer their users operations or techniques for redefining data structures without having to dump and reload the entire database. Adding a column to a table or a field to a record is the type of schema manipulation an organization may need as it expands the kinds of data that it collects and uses. A slight change in a law or regulation may force an agency to keep more complicated or detailed records of its work. Potential customers should consider not only the current requirements of applications sharing a database, but also the likelihood that the data structures may change with time.

While on some DBMS's the facilities for redefining a schema may consist of only a few commands, on others there may be a fully integrated data dictionary to record and control schema changes. In still other environments the data dictionary may stand entirely alone or work with the DBMS as a separate but related product. A data dictionary can help a database administrator manage a database by identifying relations among structures, users of particular subschemas, or programs and applications that depend on specific data

structures. Both ANSI and ICST are now working to produce a standard data dictionary system. Such a tool can make manipulation of a schema safer, but of course it imposes additional costs and overhead on data management. Whether or not a data dictionary is appropriate for a particular organization depends on the nature of both the database and the database administrator's strategy for controlling the data.

5.6 Access Control

Most applications and user environments require some form of control over access to data. In many cases a DBMS will offer a subschema or view facility that enables the database administrator to restrict users to viewing only the data that are necessary for their work. The extent of this control could vary considerably from one system to the next. Potential customers must understand both application requirements and DBMS products well enough to know which of the following questions are relevant to their plans:

1. Does access control extend to the level of records or fields?

2. Does the product provide access control based on data values or only on data structures?

3. Can users be denied the use of certain operations on the data?

4. Can users update records or tables if they have access only to certain fields or columns?

5. Does the database administrator have sufficient control over access rights to tailor users' privileges to match their needs?

6. How easy is it to change access rights to reflect changes in user needs?

7. Are access rights attached to the user or to the data, and which way better suits application requirements?

Whether it is implemented through a subschema facility or through some other mechanism, access control is an important consideration for almost any organization that wants

centralized data management.

5.7 Concurrency Control

Sharing data will almost inevitably cause conflicts between users who want access to the same data at the same time [FIPS81b]. Fortunately, research on concurrency control in operating systems applies directly to problems in common databases. Anyone considering a DBMS for multiple users should make sure that the product has adequate deadlock prevention or at least detection. A system with only deadlock detection should provide a means for rolling back the transactions of all blocked processes and giving one of them the necessary priority to complete the deadlocked transaction. If the DBMS simply grinds to a halt when it detects a deadlock, users may find themselves wasting valuable time whenever they want to update the database.

The proposed NDL and RDL standards handle concurrency by defining the notion of a transaction, which is a sequence of operations (statement executions) that is atomic with respect to recovery and concurrency. A transaction terminates with commit or rollback. If it terminates with rollback, then all changes that it made to the database are canceled. If it terminates with commit, then the changes become part of the database. Committed changes cannot be canceled, and changes made to the database by a transaction cannot be perceived by other transactions until such time as that transaction terminates with commit. The execution of concurrent transactions is guaranteed to be serializable, which means that the execution of the operations of concurrent transactions produces the same effect as some serial execution of those same transactions. A serial execution is one in which each transaction executes to completion before the next transaction begins. Serializable execution of transactions implies that all read operations are reproducible within a transaction, except for changes made by the transaction itself.

5.8 Access Languages

Users of a database may demand specific languages for their applications, and the purchaser of a DBMS should certainly try to match those demands against the features of competing products. If an organization has an absolute requirement for database access from FORTRAN, COBOL, and Pascal programs, then it must make sure not only that the DBMS

allows such access, but also that the planned environment has suitable compilers for those languages. If a particular product allows access from any language that can call separately compiled procedures, but runs only under an operating system for which no one has written a COBOL compiler, then it cannot meet the organization's needs.

Besides access from traditional programming languages, users may want even higher-level languages that enable them to retrieve data quickly without going to the trouble of writing a program. Navigational systems usually offer facilities that allow ad-hoc retrieval of data for queries or reports. Relational systems generally offer a query language based on either the relational algebra or the relational calculus, and many products based on other models offer similar languages. Because of their precision and elegance, these languages appeal to sophisticated users like the people who have to select and administer a DBMS. But for the typical nonprogrammer a language based on predicate calculus is probably unsuitable. For these users many vendors offer powerful user languages and report writers that simplify access to the database. One vendor's user language reportedly offers a 90 percent savings in code and a 75 percent savings in effort relative to COBOL [DRAP81]. Though there are no candidate standards for user languages and report writers, some organizations may consider them very attractive features of a prospective DBMS. Users should be aware, though, that reliance on one vendor's reporting language for building an application means either total dependence on that vendor through the application's life cycle, or expensive rewriting in another language.

5.9 Display Features

While a wide range of access languages may be essential to some DBMS customers, it might not be enough. A number of companies offer attractive application packages that take advantage of the display features of intelligent terminals. Such packages can make database management substantially easier, especially in an environment where a number of database users have little or no programming experience. A forms processor capable of reading a screenful of data at a time can eliminate some of the syntactic errors that inevitably occur in writing a program, no matter how simple the language. Of course, such a powerful tool may require equally powerful protective mechanisms to ensure the integrity of the database.

5.10 Support, Training, and Documentation

Most of the features mentioned so far have to do with the DBMS itself. In procuring a software system as complex as a DBMS, the customer must also consider the quantity and quality of services provided by the vendor. Few, if any, user organizations can handle their own software support for a proprietary package, and training and documentation are essential. Potential customers should determine the length and coverage of vendor warranties, the cost of continued software support and system upgrades, and the skill and responsiveness of the vendor's technical support staff. They should also find out as much as possible about training courses and examine system documentation, including on-line help facilities. The quality of these customer services could greatly affect the ease of using and maintaining a DBMS.

5.11 Existing User Base

For many products as complex as a DBMS there are national and local users' groups that can offer additional support to new customers. Although frequently sponsored by the vendors themselves, these groups generally express the interests of the users. Through organization customers can often achieve effective, reliable response to their needs for bug fixes, design changes, or enhancements in future releases of a system. Furthermore, other users of a particular product may have developed similar application packages or local programs that can significantly decrease the time and cost of developing application systems for use with the DBMS. User groups offer a pool of experienced people who can benefit from each other's mistakes and successes.

5.12 Benchmarks and Prototypes

However much a customer may learn about a DBMS through discussion and reading, nothing substitutes for a good demonstration. If it is at all practical, an organization planning extensive database applications may want to develop benchmarks or prototypes to test whether a given product really does meet user requirements. For smaller systems a feature analysis or some actual experience with small prototypes may be sufficient. While building benchmark programs or constructing a prototype application may be costly, it may provide crucial performance results that otherwise would not be available until after purchase. In some cases

vendors may be willing to help build a sample database with appropriate programs to demonstrate the practical benefits of their particular product. Careful comparison of benchmark results could be the determining factor in justifying the selection of one DBMS over another. A forthcoming NBS Special Publication discusses some alternatives to benchmarking [LETM83].

6. CONCLUSION

People who are planning to use a DBMS need a systematic way to match application requirements with the features of commercial products. By defining basic structures and operations, data models give users a tool both for analyzing their requirements and for comparing alternative systems. Ideally, it would be helpful to have precise specifications of the data models underlying all commercial DBMS's. In practical terms, however, we have to rely on models whose specifications are in the public domain and are therefore accessible to any company that wants to build a conforming product. ANSC X3H2 is working on two such specifications, one for network and another for relational databases.

These two models differ not only at the level of individual structures and operations, but also in their design "philosophies." The network model provides some complex inter-record structures that can contain a lot of valuable information. Structures in the relational model are simpler, but powerful relational operations compensate for the lack of structural information. As a result of this difference in orientation, the network and relational models serve different needs. A network DBMS may be more appropriate for an organization whose applications are relatively stable and therefore require few structural changes. Relational DBMS's rely so heavily on data manipulation that they can often accommodate new, unforeseen data requirements.

While there are far more data models than the two currently in the process of standardization, most of the others are either structure-oriented like the network model or operation-oriented like the relational model. Particular vendors have in many cases developed tools or features that compensate for some of the limitations inherent in the data model underlying their products. Potential customers, however, can still use the discussions of network and relational models presented in this report as guidance in their

evaluation of other models and as a caution to recognize that every product has its advantages and its disadvantages. The difficult job is still to match the features of a proposed DBMS with the particular application requirements of the users.

7. ACKNOWLEDGMENTS

We would like to acknowledge the members of American National Standards Committee X3H2, whose careful and precise work on the specifications of the network and relational data models has provided so much of the background material for this report.

8. REFERENCES

[ANSI77] American National Standards Institute, Inc., American National Standard Code for Information Interchange, ANSI X3.4-1977, June 9, 1977.

[ASTR75] Astrahan, M. M. and D. D. Chamberlin, "Implementation of a Structured English Query Language", in Communications of the ACM 18:10 (1975), pp. 580-587.

[BACH64] Bachman, C.W. and S.B. Williams, "A General Purpose Programming System for Random Access Memories," Proc. Fall Joint Computer Conf., October 1964, 26, pp.411-422.

[CHEN76] Chen, Peter P., "The Entity Relationship Model -- Toward a Unified View of Data", in ACM Transactions on Database Systems (March 1976), pp. 9-36.

[CHEN79] Chen, Peter P. Proceedings of the International Conference on Entity-Relationship Approach to Systems Analysis and Design. Los Angeles, CA, December 10-12, 1979.

[CHEN81] Chen, Peter P. Entity-Relationship Approach to Information Modeling and Analysis: Proceedings of the Second International Conference on Entity-Relationship Approach. Washington, DC, October 12-14, 1981.

[CHEN82] Chen, Peter P., I. Chung, and D. Perry. A Logical Database Design Framework. GCR 82-390, National Bureau of Standards, 1982.

[CODA81] CODASYL DDLC JOD 1978, Canadian Government Publishing Centre, Ottawa, Ontario K1A0S9.

[CODD70] E.F. Codd, "A Relational Model of Data for Large Shared Data Banks," Communications of the ACM, Vol. 13, Number 6, June 1970, pp. 377-87.

[DRAP81] Draper, Jesse M. Costs and Benefits of Database Management: Federal Experience. NBS Special Publication 500-84, National Bureau of Standards, 1981, p. 27.

[FIPS79] Guideline for Automatic Data Processing Risk Analysis. Federal Information Processing Standards (FIPS) Publication 65, National Bureau of Standards, August 1, 1979.

[FIPS81a] Guidelines for ADP Contingency Planning. Federal Information Processing Standards (FIPS) Publication 87, National Bureau of Standards, March 27, 1981.

[FIPS81b] Guideline on Integrity Assurance and Control in Database Administration. Federal Information Processing Standards (FIPS) Publication 88, National Bureau of Standards, August 14, 1981.

[ISO73] International Organization for Standardization, "7-bit coded character set for information processing interchange," ISO 646-1973(E), 1st edition, 1973-07-01.

[ISO82] International Organization for Standardization, "Information processing -- ISO 7-bit and 8-bit coded character sets -- Code extension techniques," ISO 2022-1982(E), 2nd edition, 1982-12-15.

[JOHN82] Johnson, Rowland, "A Data Model for Integrating Statistical Interpretations", in Proceedings of the First LBL Workshop on Statistical Database Management (March 1982), pp. 176-189.

[LETM83] Letmanyi, Helen. Alternatives to Benchmarking for Evaluating Computer Systems for Federal Agency Procurement. NBS Special Publication 500-xxx, National Bureau of Standards, forthcoming.

[PERS81] "List of Micro DBMS Software Producers," Personal Computing, February 1981, p. 30.

[SHIP81] Shipman, David, "The Functional Data Model and the Data Language DAPLEX", in ACM Transactions on Database Management Systems (March 1981), pp. 140-173.

[SOFT82] "DBMS Focus Report: Sampling of Systems," Software News, February 2, 1982, p. 30.

[SPAR77] American National Standards Institute, Inc., The ANSI/X3/SPARC DBMS Framework: Report of the Study Group on Data Base Management Systems, SPARC/77-132, October 7, 1977.

[STON76] Stonebraker, M., E. Wong, P. Kreps, and G. Held, "The Design and Implementation of INGRES", in ACM Transactions on Database Systems 1:3 (1976), pp. 189-222.

[X3H283a] "(Draft Proposed) Network Database Language," American National Standards Institute technical committee X3H2 document X3H2-83-151, dated August, 1983.

[X3H283b] "(Draft Proposed) Relational Database Language," American National Standards Institute technical committee X3H2 document X3H2-83-152, dated August, 1983.

[X3J483] "(Draft Proposed) COBOL Network Database Interface with X3H2 Network Database Language," American National Standards Institute technical committee X3J4 Working Document (X3J4.3-16-14), dated July 1, 1983.

NBS-114A (REV. 2-80)

U.S. DEPT. OF COMM. BIBLIOGRAPHIC DATA SHEET (See instructions)	1. PUBLICATION OR REPORT NO. NBS SP 500-108	2. Performing Organ. Report No.	3. Publication Date January 1984

4. TITLE AND SUBTITLE

Computer Science and Technology:

Guide on Data Models in the Selection and Use of Database Management Systems

5. AUTHOR(S)

Leonard J. Gallagher and Jesse M. Draper

6. PERFORMING ORGANIZATION (If joint or other than NBS, see instructions)

NATIONAL BUREAU OF STANDARDS
DEPARTMENT OF COMMERCE
WASHINGTON, D.C. 20234

7. Contract/Grant No.

8. Type of Report & Period Covered

Final

9. SPONSORING ORGANIZATION NAME AND COMPLETE ADDRESS (Street, City, State, ZIP)

Same as item 6.

10. SUPPLEMENTARY NOTES

Library of Congress Catalog Card Number: 83-600630

☐ Document describes a computer program; SF-185, FIPS Software Summary, is attached.

11. ABSTRACT (A 200-word or less factual summary of most significant information. If document includes a significant bibliography or literature survey, mention it here)

Selecting a database management system involves matching users' requirements and the capabilities of available products. One way to simplify this task is to define data models identifying both data structures and the operations on those structures. In the past every commercial product has implemented its own data model. Now technical committee X3H2 of the American National Standards Institute is working on specifications for two models that are similar but not identical to many existing products. The network model is a structure-oriented model that is especially suitable for databases with static structures and a high volume of record-at-a-time processing. The relational model depends more heavily on operations than structures and thus provides the flexibility to handle dynamic databases. Examples written in the draft Network Database Language and the Relational Database Language demonstrate that both models can answer complex queries in a straightforward manner.

In addition to the issue of data models, prospective buyers of database software need to consider features that affect daily operations. Existing hardware and operating systems sometimes limit the choice to a few commercial products. Systems also vary widely in their facilities for backup and recovery, bulk loading, schema manipulation, concurrency control, and report writers.

12. KEY WORDS (Six to twelve entries; alphabetical order; capitalize only proper names; and separate key words by semicolons)

computer languages; computer software standards; DBMS; data management; data models; database management systems; network databases; relational databases; system selection.

13. AVAILABILITY

☒ Unlimited

☐ For Official Distribution. Do Not Release to NTIS

☒ Order From Superintendent of Documents, U.S. Government Printing Office, Washington, D.C. 20402.

☐ Order From National Technical Information Service (NTIS), Springfield, VA. 22161

14. NO. OF PRINTED PAGES

71

15. Price

ANNOUNCEMENT OF NEW PUBLICATIONS ON COMPUTER SCIENCE & TECHNOLOGY

Superintendent of Documents,
Government Printing Office,
Washington, DC 20402

Dear Sir:

Please add my name to the announcement list of new publications to be issued in the series: National Bureau of Standards Special Publication 500-.

Name _____

Company _____

Address _____

City _____ State _____ Zip Code _____

(Notification key N-503)

NBS TECHNICAL PUBLICATIONS

PERIODICALS

JOURNAL OF RESEARCH—The Journal of Research of the National Bureau of Standards reports NBS research and development in those disciplines of the physical and engineering sciences in which the Bureau is active. These include physics, chemistry, engineering, mathematics, and computer sciences. Papers cover a broad range of subjects, with major emphasis on measurement methodology and the basic technology underlying standardization. Also included from time to time are survey articles on topics closely related to the Bureau's technical and scientific programs. As a special service to subscribers each issue contains complete citations to all recent Bureau publications in both NBS and non-NBS media. Issued six times a year. Annual subscription: domestic $18; foreign $22.50. Single copy, $5.50 domestic; $6.90 foreign.

NONPERIODICALS

Monographs—Major contributions to the technical literature on various subjects related to the Bureau's scientific and technical activities.

Handbooks—Recommended codes of engineering and industrial practice (including safety codes) developed in cooperation with interested industries, professional organizations, and regulatory bodies.

Special Publications—Include proceedings of conferences sponsored by NBS, NBS annual reports, and other special publications appropriate to this grouping such as wall charts, pocket cards, and bibliographies.

Applied Mathematics Series—Mathematical tables, manuals, and studies of special interest to physicists, engineers, chemists, biologists, mathematicians, computer programmers, and others engaged in scientific and technical work.

National Standard Reference Data Series—Provides quantitative data on the physical and chemical properties of materials, compiled from the world's literature and critically evaluated. Developed under a worldwide program coordinated by NBS under the authority of the National Standard Data Act (Public Law 90-396).

NOTE: The principal publication outlet for the foregoing data is the Journal of Physical and Chemical Reference Data (JPCRD) published quarterly for NBS by the American Chemical Society (ACS) and the American Institute of Physics (AIP). Subscriptions, reprints, and supplements available from ACS, 1155 Sixteenth St., NW, Washington, DC 20056.

Building Science Series—Disseminates technical information developed at the Bureau on building materials, components, systems, and whole structures. The series presents research results, test methods, and performance criteria related to the structural and environmental functions and the durability and safety characteristics of building elements and systems.

Technical Notes—Studies or reports which are complete in themselves but restrictive in their treatment of a subject. Analogous to monographs but not so comprehensive in scope or definitive in treatment of the subject area. Often serve as a vehicle for final reports of work performed at NBS under the sponsorship of other government agencies.

Voluntary Product Standards—Developed under procedures published by the Department of Commerce in Part 10, Title 15, of the Code of Federal Regulations. The standards establish nationally recognized requirements for products, and provide all concerned interests with a basis for common understanding of the characteristics of the products. NBS administers this program as a supplement to the activities of the private sector standardizing organizations.

Consumer Information Series—Practical information, based on NBS research and experience, covering areas of interest to the consumer. Easily understandable language and illustrations provide useful background knowledge for shopping in today's technological marketplace.

Order the **above** *NBS publications from: Superintendent of Documents, Government Printing Office, Washington, DC 20402.*

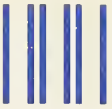
Order the **following** *NBS publications—FIPS and NBSIR's—from the National Technical Information Service, Springfield, VA 22161.*

Federal Information Processing Standards Publications (FIPS PUB)—Publications in this series collectively constitute the Federal Information Processing Standards Register. The Register serves as the official source of information in the Federal Government regarding standards issued by NBS pursuant to the Federal Property and Administrative Services Act of 1949 as amended, Public Law 89-306 (79 Stat. 1127), and as implemented by Executive Order 11717 (38 FR 12315, dated May 11, 1973) and Part 6 of Title 15 CFR (Code of Federal Regulations).

NBS Interagency Reports (NBSIR)—A special series of interim or final reports on work performed by NBS for outside sponsors (both government and non-government). In general, initial distribution is handled by the sponsor; public distribution is by the National Technical Information Service , Springfield, VA 22161, in paper copy or microfiche form.

U.S. Department of Commerce
National Bureau of Standards

Washington, D.C. 20234
Official Business
Penalty for Private Use $300



POSTAGE AND FEES PAID
U.S. DEPARTMENT OF COMMERCE
COM-215

SPECIAL FOURTH-CLASS RATE
BOOK