Evolution of Data-Base Management Systems*

JAMES P. FRY

Graduate School of Business Administration, University of Michigan, Ann Arbor, Michigan 48109

EDGAR H. SIBLEY

Department of Information Systems Management, University of Maryland, College Park, Maryland 20742, and National Bureau of Standards, Washington, D.C. 20234

This paper deals with the history and definitions common to data-base technology. It delimits the objectives of data-base management systems, discusses important concepts, and defines terminology for use by other papers in this issue, traces the development of data-base systems methodology, gives a uniform example, and presents some trends and issues.

Keywords and Phrases: data base, data-base management, data definition, data manipulation, generalized processing, data model, data independence, distributed data base, data-base machines, data dictionary

CR Categories: 3.51, 4.33, 4.34

1. GENERALIZED PROCESSING

A data-base management system (DBMS) is a generalized tool for manipulating large data bases; it is made available through special software for the interrogation, maintenance, and analysis of data. Its interfaces generally provide a broad range of language to aid all users—from clerk to data administrator.

DBMS technology can be traced back to the late fifties, when authors such as McGee [G1 and G2][1] discussed the success of "generalized" routines. These routines were capable of sorting any file regardless of its data content (the user merely supplying parameters to direct the major elements of the sorting process); those authors then proposed that these ideas be extended into other data-processing areas, such as file maintenance and report generation. This *generalized processing* entails the building of special data functions which perform frequently used, common, and repetitive data-processing tasks. But such generality cannot be accomplished without cost. The price of generalized processing is a reduction in operating efficiency, often through interpretive processing, or a necessary increase in resources such as hardware capacity. The success of generalized processing (and consequently of generalized data-base technology) thus becomes an issue of cost tradeoff.

Hardware improvements developed over the past two decades have effected significant decreases in price/performance ratio, thereby tending to offset operational inefficiency and to emphasize the cost of application and software development. The

* This work is sponsored in part by the National Science Foundation Grant GJ 41831.

[1] *Editor's Note:* See page 35 for the key to the classifiation system used for references cited in this paper.

CONTENTS

benefits of a generalized approach can thus be summarized as the elimination of program duplication (frequently found in computing systems), and the amortization of the one-time development costs over many applications of the program.

In cases where a particular data-processing application cannot be parameterized, the usual recourse is to develop high-level languages, which are themselves a form of parameterized generalized processing, albeit with very special parameters. For example, the development of high-level interrogation languages for ad hoc requests has broadened the user access to data by providing a simple and, it is hoped, easy-to-use interface. Such an approach allows the inquirer to use a language similar to everyday English, rather than requiring him to write a program in an artificial language. Generalized data-processing techniques have evolved into a class of sophisticated, generalized software systems, one of which is the data-base management system. The reader should carefully distinguish between the terms DBMS and "data management." The latter has been used by the government to designate an administrative function, by some hardware vendors to designate their access methods, and by some software vendors to designate and embellish comprehensive packaged systems.

2. OBJECTIVES OF DATA-BASE MANAGEMENT

The Guest Editor's Introduction to this issue of COMPUTING SURVEYS discussed the concepts of data-base technology and introduced some of its objectives:

- to make an integrated collection of data available to a wide variety of users;
- to provide for quality and integrity of the data;
- to insure retention of privacy through security measures within the system; and
- to allow centralized control of the data base, which is necessary for efficient data administration.

To this we add the objective of "data independence," a term to be defined later [see page 12] in this paper. This section will deal with each of the stated objectives, relating them to the overall functional architecture of the DBMS.

While various "views of data" (the principal topic of this issue of COMPUTING SURVEYS) are important to the user interface, the requirements for quality, integrity, security, and control have far-reaching effects on the overall cost, accessibility, and per-

formance of the system. Although it is possible to add functional capabilities to an existing system, the cost of retrofitting is often prohibitive, and the post-design addition may adversely affect the system performance. Although quality, security, and control factors are given relatively scant treatment in other papers in this issue of SURVEYS, it should not be inferred that these are unimportant. In fact, the consequences of excellent or poor satisfaction of these needs may make or break a working system.

Data Availability

Everest [G12] states that the major objective of a DBMS is to make data sharing possible. This implies that the data base as well as programs, processes, and simulation models are available to a wide range of users, from the chief executive to the foreman (Everest and Sibley [G8]). Such sharing of data reduces its average cost because the community pays for the data, while individual users pay only for their share. However, under these circumstances the data cannot "belong" to any individual, program, or department; rather, it belongs to the organization as a whole.

What, then, is the overall cost of data? One way to answer this question is by observing data entry. Keypunching and verifying, or other types of data entry involving human keystroking, tend to cost about 50¢ per thousand characters input. Thus, if the average-sized file is two million characters (a figure representative of much of today's industry and government), it costs $1000 to input each average-sized file. Under certain conditions the cost of collecting data could be substantially higher, e.g., when the data must be collected by telemetry, or in long and complicated experiments.

Another expense is associated with the *lack* of data, the so-called "lost opportunity cost." If data is not available when an important decision is to be made, or if duplicate but irreconcilable data exists, an ad hoc and possibly wrong decision results. Nolan [A4] gives a scenario of a typical business where a manager knew that data existed, but some of it had been produced on a different machine and some had incompatible formats (different structures on different tapes). Moreover, none of the data definitions were easily available. The manager who needed the data for important predictions was unable to obtain answers in a reasonable amount of time.

There are two important mechanisms for making data available: the "data definition" and the "data dictionary." A data definition is a more sophisticated version of a DATA DIVISION in COBOL, or a FORMAT statement in FORTRAN; however, a data definition is supplied *outside* the user program or query and must be attached to it in some way. The *data definition* (as specified by a data administrator) generally consists of a statement of the names of elements, their properties (such as character or numerical type), and their relationship to other elements (including complex groupings) which make up the data base. The data definition of a specific data base is often called a *schema*.

When the data definition function is centralized (which is necessary to achieve the objectives of DBMS), control of the data-base schema is shifted from the programmer to the data administrator [A1]. The programmer or the ad hoc user of a query language is no longer able to control many of the physical and logical relationships. While this restricts the programmer to some extent, it means that all programs use the same definition; thus any new program can retrieve or update data as easily as any other. Furthermore, greater data definition capabilities are provided, the storage and retrieval mechanisms are hidden from the program, the formats cannot be lost, and the programmer's task is simpler.

Centralized data definition facilitates the control of data duplication, which generally entails some storage inefficiency. However, not all duplication of data is bad; a controled duplication may be necessary to allow special classes of users to obtain especially fast responses without penalizing quality for other users.

The data definition facility is inherent to all DBMS. Without it, the data base is owned by its programs, difficult to share,

and generally impossible to control. This, then, is the cornerstone of data-base management systems.

Whereas the data definition facility is the data administrator's control point, the *data dictionary* [D1] provides the means of broadcasting definitions to the user community. The data dictionary is the version of the data definition that is readable by humans. It provides a narrative explanation of the meaning of the data name, its format, etc., thus giving the user a precise definition of terms; e.g., the name TODAYS-DATE may be defined narratively and stated to be stored in ANSI standard format as Year: Month: Day.

Within the past five years a number of data dictionary packages have appeared on the market [D2]. Some of these are an integral part of the data definition function, while others provide an interface to multiple DBMS, and still others are stand-alone packages.

The dictionary will normally perform some, if not all, of the following functions: storage of the definition, response to interrogation, generation of data definition for the DMBS, maintenance of statistics on use, generation of procedures for data validation, and aid in security enforcement. Obviously, storage of the data definitions in the dictionary is obligatory.

The dictionary wiil normally be able to either provide formatted dictionaries (on request) or respond to a simple query for a data entry, or to do both. This facility allows ad hoc users to browse through the definitions (on- or off-line) to determine correct data names.

In some dictionary systems, especially those that augment a DBMS, the data administrator can invoke a data definition generator. This allows the administrator to pick names of elements from the dictionary, group them, and then produce a new data definition.

The dictionary may be both a collector for, and a repository of statistics on DBMS usage. These statistics can be utilized to improve the efficiency of the DBMS by regrouping elements for better accessing.

The dictionary may contain information on techniques for validation of particular elements, and the data validation statements can be used to generate procedures for input editing or other quality checking.

The data dictionary is extremely important as part of the DBMS security mechanism. If an adversary knows you are gathering data, that adversary has already violated your security. For this reason, the data dictionary should be as secure as the DBMS. Furthermore, if security requirements are retained in the dictionary they can be automaticaliy checked (and special procedures can be invoked) every time a data definition is produced for the DBMS. This would improve security monitoring.

Data Quality

Perhaps the most neglected objective of DBMS is the maintenance of quality. Problems relating to the quality of data and the integrity of systems and data go hand-in-hand. Data may have poor quality because it was:

- never any good (GIGO—garbage in, garbage out);
- altered by human error;
- altered by a program with a bug;
- altered by a machine error; or
- destroyed by a major catastrophe (e.g., a mechanical failure of a disk).

Maintenance of quality involves the detection of error, determination of how the error occurred (with preventive action to avoid repetition of the error), and correction of the erroneous data. These operations entail precautionary measures and additional software functions within the data-base management system. The prevention and correction of the five listed causes of error will now be briefly discussed.

In dealing with normal data-processing applications, the programmer is faced with a great deal of input validation. A survey by the authors showed that about 40% of the PROCEDURE divisions of present-day industrial COBOL programs consists of error-checking statements. If the validation requirements can be defined at data definition time, then error checks may be applied automatically by the system at input, update, manipulation, or output of data, depending on the needs specified by the data adminis-

trator. Many current DBMS allow validation. Some have a check mechanism which ensures that the values conform to the stated PICTURE (like COBOL); they also check that the value is within the defined range, or that it is one of a predefined set. If a system is to support such techniques it must have special clauses in the data definition language (DDL), as well as a series of procedures to be invoked on error detection.

A second cause of poor data is human or program error. Little can be done to prevent such errors unless they contravene some validation rule, and their discovery normally involves human knowledge. The cause, however, may be detected by referring to the "audit trail." An *audit trail* is a log in some journal of all changes made to the data base. When a change is to be made, there are two important objects: the original data and the changed data. When logged, these objects are termed "before" and "after" images. Generally, these images contain the data, time, and name of the procedure causing the change. They may also record the name of the person who initiates the procedure. A *quality audit* is an attempt to determine, through examination of the before and after images, who or what procedure changed the data value. A quality audit may find that some user promulgates many errors, whereupon the data administrator may request that the user take more care (or a course in better techniques). If, however, the error appears to have been generated by some operational program, a programmer may be called in to debug it.

Sometimes an error will be detected after a procedure is partially completed. In this case, as well as when a user makes a mistake, it is often necessary to "back-out" the procedure or "back-up" the data base. This is a process of reinstating the data that has been incorrectly updated. Many data-base management systems provide an automatic facility for reinstatement, achieved by reading the before images from the audit trail and replacing any updated data with its prechange value.

Poor quality data can also be generated by an unpredicted disaster. The process of recovering from a permanent hardware failure, or of restarting after a minor error generally involves the use of the audit trail. In modern operating systems a *restart* facility is often provided. Normally, in order to restart the DBMS after a failure which does not involve physical damage to the storage devices, a "checkpoint" facility is used. A *checkpoint* is a snapshot of the entire machine condition (CPU, memory, etc.) recorded on the log. This entry presents a known condition of the entire system.

A checkpoint may be either taken by the computer operator or automatically initiated by the DBMS. Usually the latter method is triggered by a procedure which keeps count of the number of transactions processed and then initiates the checkpoint when a predefined value is exceeded. The problem with such facilities is that they often need a quiescent system, i.e., one in which the transactions are being held in queues or have been completed. This "freeze" operation may take some time. Unstarted procedures are held until the checkpoint process has been completed, causing a delay which can lead to dissatisfied users.

After any major error it is possible to back-up to the latest checkpoint on the log and then move forward along the log, replacing updated (after) images of completed transactions or reinitiating unfinished transactions. Recovery can be a complicated process, and many current data-base management systems rely substantially on the capabilities of the underlying operating system to perform the function.

Sometimes a major storage failure (e.g., a disk crash) requires replacement of hardware and total reloading of the (possibly very large) data base. It is not unusual to find commercial and governmental data bases with over one billion characters. A sequential load of such a large data base may take two to six hours on present-day computers. The reload is from a *data-base dump*, that is, from a copy taken at some time in the past (assuming possible failure of the original). A data-base dump only represents the status of the data base at a certain time, and any updating performed subsequent to that time must be replicated by using the log. Many current systems use such techniques, although some still rely on reinitiating and

reprocessing the logged update transactions. This procedure tends to be very slow.

The quality and integrity of data depend on input-validation techniques in the original data definition, logging of data-base changes, periodic snapshots of the entire machine status, and total or incremental data-base dumping. These operations require additional software in the data-base management system, both for initiation of the protective feature and for its utilization to reconstitute a good data base. However, they entail an overhead expense which adds to the normal running cost.

Privacy and Security

The third major objective of data-base management systems is privacy—the need to protect the data base from inadvertent access or unauthorized disclosure. Privacy is generally achieved through some *security* mechanism, such as passwords or privacy keys. However, problems worsen when control of the system is decentralized, e.g., in distributed data bases, where the flow of data may overstep local jurisdictions or cross state lines.

Who has the responsibility for the privacy of transmitted data? When data requested by someone with a "need to know" is put into a nonsecure data base and subsequently disseminated, privacy has been violated. One solution to this problem is to pass the privacy requirements along with the data, which is an expensive, but necessary addition. The receiving system must then retain and enforce the original privacy requirements.

Security audits, another application of the audit trail, are achieved by logging access (by people and programs) to any secure information. These mechanisms allow a security officer to determine *who* has been accessing *what* data *under what conditions*, thereby monitoring possible leakage and preventing any threat to privacy. Much of this technology is, however, still in its infancy.

Management Control

The need for management control is central to the objectives of data-base management. It includes the establishment of the data administration function and the design of effective data bases. Data administration currently uses primitive tools; a discussion of them would be beyond the scope of this paper (see [A1, 2, and 3]). However, it is important to note that data-base design involves tradeoffs, because users may have quite incompatible requirements. As an example, one group may require very rapid response to ad hoc requests, while another requires long and complicated updating with good security and quality control of the data. The implementation of a system responsive to the first need may suggest a storage technique quite different from that needed by the second. The only way to resolve such a conflict is to determine which user has the major need. If the requirements are equally important, a duplicate data base may be necessary—one for each class of user.

Although the installation of a data-base management system is an important step toward effective management control, today's data administrator faces a challenge: the available tools are simplistic and seldom highly effective. They involve simulation, data gathering, and selection techniques. Some new analytical methods appear promising [G3]. These methods select the "best" among several options of storage techniques, but they are usually associated with one particular DBMS rather than with several.

Data Independence

Many definitions have been offered for the term data independence, and the reader should be aware that it is often used ambiguously to define two different concepts. But first, we must define other terms. A *physical structure*[2] describes the way data values are stored within the system. Thus pointers, character representation, floating-point and integer representation, ones- or

[2] The terms *data structure* and *storage structure*, which were promulgated by the CODASYL Systems Committee [U2] can be attributed to D'Imperio [DL2]. However, in computer science, the term data structure is more closely associated with physical implementation techniques such as linked lists, stacks, ring structures, etc. To prevent ambiguity we opt for the more basic terms, logical and physical structure.

twos-complement, or sign-magnitude representation of negative integers, record blocking, and access-method are all things associated with the physical structure. The term *logical structure* describes the user's view of data. Thus, a COBOL DATA DIVISION is (mainly) a statement of logical structure; it deals with named elements and their relationships rather than with the physical implementation. A record in a COBOL program is manipulated without knowledge of the computer hardware or its access method. As an example, the data item named AUTHOR may have values FRY, SIBLEY, FRANK, TAYLOR, CHAMBERLIN, etc. Whereas the name AUTHOR is a logical phenomenon, the representation of authors is a physical phenomenon.

In the early days of DBMS, the term "physical data independence" was used. A system was said to be (physically) data independent if it could deal with different physical structures and/or different access methods, and if the user of the data base did not have to provide information on details of the structure. Thus a definition of *physical data independence* is:

> *A system is data independent if the program or ad hoc requests are relatively independent of the storage or access methods.*

Systems with physical data independence provide a discrete number of choices for implementing the physical storage of data. Other systems also allow the user to make requests with little knowledge of the logical structure of the data. Such systems, which are said to have logical data independence, may operate correctly even though the logical structure is, within reason, altered. A definition of *logical data independence* is:

> *The ability to make logical change to the data base without significantly affecting the programs which access it.*

Logical data independence has two important aspects; first, the capability of a data-base management system to support various (system or user) views of the data base, and second, the capability of the data-base management system to allow modification of these views without adversely impacting the integrity of existing applications. The latter capability is important in the restructuring function [G13], but this definition of data independence is perhaps too broad. It suggests that substantial logical change could be made without creating a need to change the programs—a difficult, if not impossible task. However, a serious attempt is being made to understand how much logical change can be made without adverse affect on the program. Some of the different models discussed in this issue of SURVEYS claim to be more data independent than others. Full data independence appears, however, to involve an understanding of *data semantics*, the formalization of the meaning of data. Research on data semantics is currently in its infancy.

3. FUNDAMENTAL CONCEPTS AND DEFINITIONS

Some important ideas were introduced when we discussed the basic objectives of DBMS. This section presents further concepts and definitions.

Unfortunately, our language is rich in its words and semantics about data. Entity, item, name, element, value, instance, and occurrence (to name a few) come ready-equipped with meaning, yet they are used in different ways. We must be precise, and are thus forced to make exact definitions for these words which we must use consistently.

Elements of Logical Structure

The starting point is to define the object of the discourse, the entity, and the process of its definition, which is a modeling process. A human being is constantly "modeling" information—a baby sees an animal and says "dog" (though it may be a horse). The process of modeling information as data often involves trial-and-error. First, information needs are determined, next data (and processes) are structured to satisfy the needs, and then data is restructured because of changes in the need or necessary improvements to the model.

The principal construct in the data structuring process is the entity:

> An information system deals with objects and events in the real world that are of interest. These real objects and events, called *entities*, are represented in the system by data. Information about

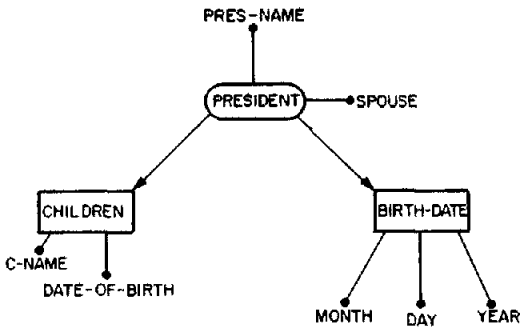

FIGURE 1. The PRESIDENT entity.

a particular entity is in the form of "values" which describe quantitatively and/or qualitatively a set of attributes that have significance in the system [M1].

Thus the goal of the data structuring process involves the collection of *data* (a set of facts capable of being recorded) about some identifiable entities that convey *information* (i.e., meaning to humans). The repository for the data is the *data base*. A data base is described in terms of its logical structure; this provides a template for the data instances which constitute the data base.

Data about an entity is generally recorded in terms of some attribute(s) that describes the entity. In the description of the data base, separate attributes are called *elementary items* or, in brief, *items*, while the collection of elementary items is termed a *repeating group* or *group*. For example, the entity PRESIDENT may be described in terms of items PRES-NAME, SPOUSE, the group BIRTH-DATE, and the repeating group CHILDREN. The group BIRTH-DATE is made up of MONTH, DAY, and YEAR, while the repeating group CHILDREN is made up of C-NAME and DATE OF-BIRTH. There may be zero or many repetitions of the CHILDREN group. The definition of the PRESIDENT entity is illustrated in Figure 1. This represents, of course, only one possible model of a definition of the PRESIDENT entity. Another user may have a different viewpoint, and need to add PARTY and STATE to the model. Thus the data base depends on the (assumed) usage, and the model may need to be changed during the life of the system.

It is possible to describe a "data model" of an entity in a formal fashion using a set-theoretic notation where:

- all repeating groups are enclosed in { } to represent the fact that they may repeat as many times as necessary, and
- all ordered sets or n-tuples are enclosed in ⟨ ⟩ to show that the order of the items is important.

In this way, the entity PRESIDENT may be defined as in Display 1 below.

An *instance* or *occurrence* of an entity is a set of values for each of the items of the entity definition. Any repeating group in an entity has an occurrence which consists of one value for each of its items. However, there may be potentially zero or an unlimited number of occurrences of a repeating group in the occurence of the entity. Naturally, each element value should be valid, i.e., it should conform to the rules and must be one of the possible members of the set of allowable values.

If the names of the presidents of the United States are the value set (or range) of the domain named PRES-NAME, then we have the value set in Display 2 below and can construct one instance of PRESIDENT as:

⟨FORD, BETTY, ⟨7, 14, 1913⟩,
{⟨SUSAN, ⟨12, 10, 57⟩⟩, ⟨JOHN, ⟨09, 03, 52⟩⟩,
⟨STEPHEN, ⟨12, 21, 55⟩⟩, ⟨MICHAEL, ⟨09, 17, 50⟩⟩⟩}⟩.

For almost any real-world situation there are many entities of interest which are related in some fashion. In a Presidential Information System there will be entities such as PRESIDENT, CONGRESS, ELECTION, STATE, and ADMINISTRATION, all of which are interrelated; for example,

Display 1:

PRESIDENT = ⟨PRES-NAME, SPOUSE, BIRTH-DATE, {CHILDREN}⟩
where BIRTH-DATE = ⟨MONTH, DAY, YEAR⟩
and CHILDREN = ⟨C-NAME, DATE-OF-BIRTH⟩

Display 2: Value Set of PRES-NAME = {FORD, NIXON, JOHNSON, KENNEDY, . . .},

PRESIDENTs "WIN" ELECTIONs, STATEs are admitted during a President's ADMINISTRATION, and PRESIDENT(s) serve with CONGRESS(es). A relationship may therefore exist between the instances of two entities. Typically, there are at least three types of relationships:

- *One-to-one:* some of the Presidents are first native sons of some states; for example, Washington (one President) was the first native son of Virginia (one state).
- *One-to-many:* during an Administration several STATES may be admitted, but a state is not admitted more than once in different Administrations.
- *Many-to-many:* a President serves with many Congresses, and a Congress may serve under many PRESIDENTs.

More on this topic is discussed in other articles in this issue of SURVEYS, but before going further, the reader should note that the following statements have different meanings.

- "The relationship named A exists between two entities, B and C"; and
- "Two instances P_1 and Q_1 of entities P and Q, respectively, are related by A".

The first is a logical statement. It states logical relationships that may occur between two entities; for example, Presidents (B) win (won) (A) Elections (C). The second statement refers to current values of data in the data base; for example, NIXON (Q_1), a PRESIDENT (Q), wins (won) (A) the 1972(P_1) ELECTION (P).

Relationships may be explicit or implicit. The entities may be joined by some naming convention (such as WIN), or the relationship may be implied (as in the example of PRESIDENT with the repeating group CHILDREN).

Generally, the instances of certain items in a group are in one-to-one correspondence to an instance of the entity. For example, the year of an election may uniquely identify a presidential election, or the congress number may uniquely define a Congress. These items are called *identifiers* or *candidate keys*. A key may be considered either a logical or a physical phenomenon: the key may be used to identify an entity (logical), or it may cause the system to sort the set of instances of entities into an order based on the value of the key (physical). In this issue of SURVEYS, key will be considered a logical conpect, but note that this definition allows "sort by key" as a physical attribute of the data base.

The discussion of entities, items, and groups has involved logical structure. The definition of this structure (a schema) requires some formal language, which is termed a *data definition language* (DDL). This language may be formatted, like a COBOL DATA DIVISION, or be relatively free-form. The following three articles in this Special Issue give specific examples of DDL usage.

Introduction to Data Models

The evolving field of data models is often hotly debated. Proponents of each model point out its advantages, but so far there is no concensus as to the best version. In reality, there is a spectrum of data models ranging from the COBOL-like "flat file" (single entity model) to the complex extended-set model.

Since COBOL, the most widely used language today, has a DATA DIVISION with data definition capabilities, it represents a good starting point for the discussion of data models. Though limited, this data definition capability allows the group (termed a RECORD in COBOL) to be defined as an 01 level, followed by the items, groups, and repeating groups at other levels. The PRESIDENT entity, discussed previously, is shown in Figure 2.

In COBOL, each item is formatted by de-

```
01 PRESIDENT.
   02 PRES-NAME PICTURE  X(20)...
   02 SPOUSE PIC X(10)...
   02 BIRTH-DATE...
      03 MONTH...
      03 DAY PIC...
      03 YEAR...
   02 CHILDREN OCCURS 0 TO N TIMES...
      03 C-NAME...
      03 DATE-OF-BIRTH...
```

FIGURE 2. A COBOL-like definition for PRESIDENT.

fining a PICTURE. Thus PRES-NAME is shown as 20 characters long, SPOUSE is 10 characters long, while DAY is two numerics.

The COBOL definition deals with one entity (defined at the 01 level), but a COBOL structure may also be termed "contained" because the groups BIRTH-DATE and CHILDREN are contained within the PRESIDENT entity (see Figure 3). There may be many levels of containment of groups within groups. There is no semantic reason why groups shown as contained in the PRESIDENT entity should not, by some other model or user, be considered separate entities; i.e., BIRTH-DATE and CHILDREN might each be entities. The relationship between PRESIDENT, BIRTH-DATE, and CHILDREN entities may, however, be constrained because the two latter are contained entities that are not really separate, but rather, are "owned" by the PRESIDENT entity. Such a model is said to be "hierarchical." Thus a *hierarchy* of entities involves a superior entity and one or more inferior entities, each of which may participate as superior entities at a third level, etc. A hierarchy represents a "tree," "bush," or "fan-out" of entities all related by a family-tree-like relationship (with no sons shared by different fathers). The topmost level of the hierarchy is termed the *entry* or *root*—terms arising because the "way in" to the entity is its entry, or (when stood on its head) it is the "root" of the tree.

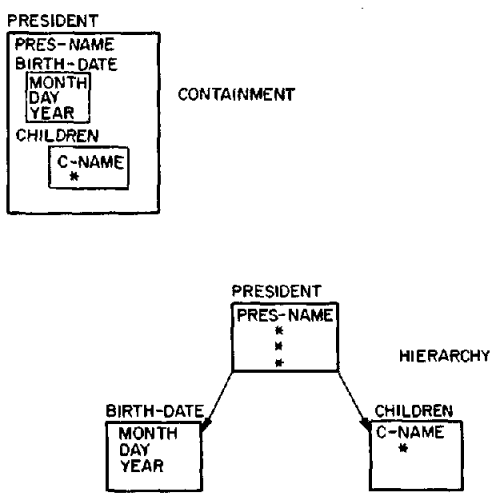

FIGURE 3. The PRESIDENT entity as contained and hierarchical structures.

Logically, containment and hierarchical representations are equivalent; however, the physical implementation of such systems causes differences in the way they are manipulated. (Hierarchic systems are discussed in this issue by Tsichritzis and Lochovsky [see page 105].) The hierarchic model has a 1 to N relationship between an owner and member entity; e.g., for one PRESIDENT there may be many (or no) CHILDREN. It also has two constraints: no member entity can be shared by the owner entities, and no entity at a lower level may own a member at a higher level in the hierarchy (assuming the words "lower" and "higher" refer to the position down a page, with the root at the top). The second constraint really follows from the first, but it has important effects.

If we first relax the multiple ownership constraint, it is possible to have the same member entity participating in two different relationships with a single owner entity. This requires a means of distinguishing the relationships. As an example, the relationships between PRESIDENT and STATE may be both ADMITTED-DURING-ADMINISTRATION and NATIVE-SON: a President may be in office when one or more STATE(s) are admitted, and one state may have zero or several native sons as Presidents. This problem can be resolved by labeling the arcs (showing the relationships between the entities) with the name of the relationship, as shown in Figure 4.

The second constraint must be relaxed carefully. Most *graphical* or *network* models still retain one constraint: that no entity may participate as both owner and member in the same relationship. This may appear unfortunate or unnecessary; after all, PEOPLE do EMPLOY other PEOPLE, and some PEOPLE are PARENTS-OF other PEOPLE. However, by careful design this problem can be resolved. Discussion of this and concepts of the general network model is given in the paper by Taylor and Frank [see page 67].

At the more theoretical end of the spectrum is the class of data models based on

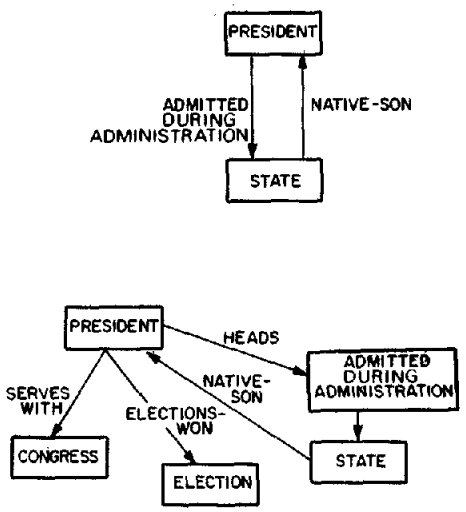

FIGURE 4. Some network structure examples.

mathematics, especially on set theory:

- relational,
- binary association, and
- extended set models.

The *relational* model [M3] deals with entities which have no containment. Thus each entity is made up only of items. The notation introduced earlier can be used to define the same three entities (two of them unchanged):

PRESIDENT = ⟨PRES-NAME, SPOUSE⟩
BIRTH-DATE = ⟨MONTH, DAY, YEAR⟩
CHILDREN = ⟨C-NAME, DATE-OF-BIRTH⟩.

But there are now no links between the three entities. These may, however, be made explicit by using the *candidate keys* to establish the relationships:

P-DATE-OF-BIRTH = ⟨*PRES-NAME, MONTH, DAY, YEAR*⟩
P-KIDS-OF = ⟨*PRES-NAME, C-NAME*⟩

assuming that the candidate keys (unique by definition) are PRES-NAME, MONTH, DAY, YEAR, and C-NAME. Another method is to link the entities implicitly by passing the owner candidate key (PRES-NAME) into the dependent entities:

PR-BIRTH-DATE = ⟨*PRES-NAME*, MONTH, DAY, YEAR⟩
PR-CHILDREN = ⟨*PRES-NAME*, C-NAME, DATE-OF-BIRTH⟩.

The instances of a group are often called *n-tuples* in the literature of relational systems, which are discussed in the paper by Chamberlin [see page 43]. Relational systems have been in use for some time at universities and research laboratories, e.g., the use of MACAIMS [Z1] and ADMINS [Z2] at MIT, and of RDMS [Z3] at General Motors Research. Some prototype systems are appearing on the market now.

The binary association model, as discussed by Senko [DL5], is part of an attempt at understanding and formalizing data semantics through the use of binary relations.

Although one of the earliest set processors was proposed in the Information Algebra [M1], Childs' set model [M2] was one of the first to be implemented, and it is also being investigated by Hardgrave [M4]. The extended set allows storage of a very wide range of ordered sets and ordinary sets, and is intended to provide maximum generality in storing relationships. However, application of these models is still in the realm of research, though one commercial system is now available [V24].

To recapitulate, information structuring (the selection of entities and specification of relationships between them) is a modeling process with little methodology, other than common sense. In order to use a DBMS, the information structure must be mapped to the logical structure of the system. The mapping is expressed in a DDL. The instances of the data (the data base) are stored by the DBMS to conform to this logical structure. A DBMS generally supports only one of the data models: relational hierarchy, or network. Since each model uses a different terminology, Table 1 attempts to equate the various terms used with the concepts that have been developed in this section.

The criteria for designing and selecting a "best" model has not yet been established—nor is it likely to be established in the near future. The user is therefore faced with two decisions: which data model to utilize (i.e., which type of DBMS), and how to structure the data using the chosen model.

Concept	Relational	Network	Hierarchic
Item	role name/domain	data item type	item/field
Item value	component	data item occurrence	value
Group	not allowed	group	group
Entity (type)	relation	record type	entry/segment type
Entity instance	tuple	record occurrence	entry/segment occurrence
Relationship	foreign key comparable underlying domains	set type	hierarchic (implied)
Relationship instance		set occurrence	assembly
Data administrator view	data model	logical structure	logical structure
Definition of data administrator view	data model definition	schema	schema
User view	data submodel		
Definition of user view	data submodel definition	subschema	subschema
Data-base subdivision		area	
Entry points	primary key	singular sets CALC records	root group root segment
Single unique/identifier	candidate key	key	sequencer (unique)

TABLE 1. COMPARATIVE TECHNOLOGY.

Mapping from the Logical to the Physical Structure

The need to create and load a data base, i.e., to make the data definition and then populate it with data, leads to the physical structure, which is the representation of data in storage. The accessing process for the data base management system is shown in somewhat oversimplified form in Figure 5. The definition of the logical structure is stored within the DBMS and associated with the request so that any logical relations may be derived. As an example, the request:

```
PRINT SPOUSE WHERE PRES-NAME = "FORD"
```

does not mention that we are dealing with a PRESIDENT entity; it is left to the DBMS to discover this fact from the logical structure. The physical mapping must have some mechanism that will determine which data to retrieve (using the key PRES-NAME if possible), and then will call the relevant operating system access method and apply any deblocking that is necessary to return the required portion of the character stream.

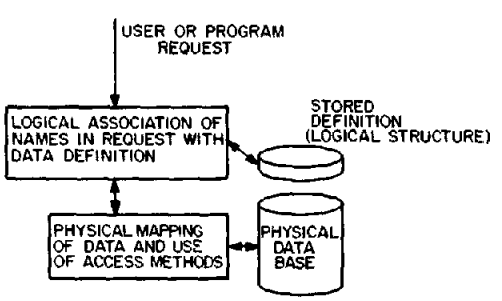

FIGURE 5. Logical and physical aspects of a DBMS.

The process of mapping from occurrences of data to their bit-string representation on disk or tape is generally system-dependent; therefore, these factors are discussed in the separate papers in this issue of SURVEYS. Most DBMS format (block and manage) the pages or records themselves, and most use the operating system access method to store and retrieve the data from secondary devices.

In fact, because most modern DBMS use the available operating system, they gen-

erally use many of its facilities. Therefore, communication management facilities, program library management, access methods, job scheduling, special program management (e.g., sorting and compiling), concurrent access prevention, checkpoint facility, etc. typically are all "adopted" by the DBMS, though some rewrite and additions may be necessary.

4. HISTORICAL PERSPECTIVES

The origin of DBMS can be traced to the data definition developments, the report generator packages, and the command-and-control systems of the fifties—a time when computers were first being used for business data processing. Many systems have been developed since the fifties (See the surveys by Minker, [U1, 4]). MITRE [U3, 8] and CODASYL (U2, 7] show numerous system implementations that have generated wide interest among users.

In 1969 Fry and Gosden [U5] analyzed several DBMS and developed a three-category taxonomy: Own Language, Forms Controled, and Procedural Language Embedded. Succinctly stated, these categories can be contrasted as follows: *Own Language Systems* (such as GIS [V16]) have a high-level, unconventional programming language; *Forms Controled Systems* (such as MARK IV [V12]) use the "fill-in-the-blank" approach, and *Procedural Language Systems* (such as I-D-S [V9]) take advantage of existing higher-level programming languages.

In 1971 the CODASYL Systems Committee [I6] observed that the most significant difference among DBMSs was the method employed in providing capabilities to the user. The Committee developed a two-category classification scheme, *Self Contained* (which included the Forms Controled category) and *Host Language*.

It is impossible to survey all systems, but it is possible to trace the evolution of the DBMS by tracing the evolution of two precursors of data base management: data definition languages and the development of generalized RPG systems.

Evolution of Data Definition Languages

One important factor in the evolution of DBMS is the development of data definition languages. They provide a facility for describing data bases that are accessed by multiple users and by diverse application programs.

Centralized Data Definition: Fifties and Sixties

Probably the first data definition facility was the COMPOOL [DL1] developed at the MIT Lincoln Laboratory for the SAGE Air Defense System in the early fifties. COMPOOL provided a mechanism for defining attributes of the SAGE data base for its hundreds of real-time programs. The COMPOOL concept was later carried over to JOVIAL [PL4] (a programming language), but some of the capability was lost when the language was implemented under a generalized operating system; the data definition became local to the language rather than global to the system.

About the same time, hardware vendors were developing programming languages for business applications: FACT [PL1] was developed by Honeywell, GECOM [PL3] by the General Electric Company, and Commercial Translator [PL2] by IBM; all provided some form of data-definition facility. GECOM and Commercial Translator provided the capability of defining intrarecord structures, and FACT offered the more advanced capability of providing inter-record hierarchical structures.

Under the aegis of CODASYL, these vendor efforts were merged into COBOL [PL5] in the late fifties. This language has a centralized DATA DIVISION which achieves the separation of the description of data from the procedures operating on it. While the DATA DIVISION initially mirrored the data as stored on tape or cards, implementors soon found themselves using different ways of physically storing data. This inherent incompatibility between physical data stored by different manufacturers becomes an important factor when data must be exchanged between two systems.

Approaches which attempt to mitigate the

data-transfer problem are the subject of recent research on the description of physical structures and the development of stored-data definition languages.

Stored-Data Definition Languages: the Seventies

One of the first efforts in this area was mounted by the CODASYL Stored-Data Definition and Translation Task Group [SL2] in 1969 with the goal of developing a language to describe stored data. At the 1970 ACM SIGFIDET (now SIGMOD) meeting, a preliminary report was made [SL3], and later reports were published in 1972 [SL5]. Notable basic research efforts in the development of these languages were reported by Smith [SL1] and Sibley and Taylor [SL4, 7] in 1971.

The Data Independent Accessing Model (DIAM) [DL3], developed by Senko and his colleagues at the IBM San Jose Research Laboratory, provides a multilevel data description capability. The description starts at the information level, structures this into a logical definition, adds encoding information, and ends with a physical description of the storage device and its logical-to-physical mapping structure. Each level provides augmentation of the description at the preceding level. Recent work by Senko [DL4,5] extends the information level in a new language called FORAL.

Thus, the single-level data description facility of the fifties, made incompatible by storage developments in the sixties, led to the recent development of stored-data description facilities in the seventies.

Development of Report Generator Systems

The development of programming languages originally allowed the user (a programmer) to define reports by giving simple definitions of the format of the lines and then writing procedures to move data into buffers prior to printing each line. Therefore, the program written to produce a complete report could consist of large numbers of statements involving expensive programming. The development of *report generators* stems from a need to produce good reports without this large programming effort. In most cases, report generators can perform complex table transformations and produce sophisticated reports from a data base. These, then, allowed the user to examine and manipulate large volumes of data, and they may be said to be a precursor, or a particular type of modern DBMS.

The Hanford/RPG Family (Figure 6)

The patriarch of today's RPG system was developed at the Hanford (Washington) operations of the Atomic Energy Commission, which was then managed by the General Electric Company. In 1956 Poland, Thompson, and Wright developed a generalized report generator [G1] (MARK I) and a generalized sort routine for the IBM 702. The capability was extended in 1957 by the development of a report and file maintenance generator (MARK II). These routines provided the basis for a joint development by several users under the SHARE organization of the 709 Package (9PAC) [W1] for the IBM 709/90.

9PAC is the principal ancestor of most commercial report generators developed since 1960. Foremost among these is the Report Program Generator (RPG) developed for the IBM 1401 in 1961; this has evolved into the RPG for the IBM System/360 and an enhanced RPG II for the IBM System/3, System/360, and several other computers [W2, 3]. Other members of the Hanford family include the COGENT systems, developed by Computer Sciences Corporation for the IBM 709 and System/360 between 1965 and 1969 [Y5], and the SERIES system [Y9].

Another system, also based on MARK II ideas, was being defined during the late fifties in a SHARE 704 Project under Fletcher Jones. This IBM 704 system, called SURGE [W4], was the predecessor of GIRLS, the partiarch of the Postley/MARK IV family.

5. DEVELOPMENT OF DBMS

The development of the data-base management systems may be divided into three somewhat overlapping periods: the early developments, prior to 1964; the establish-

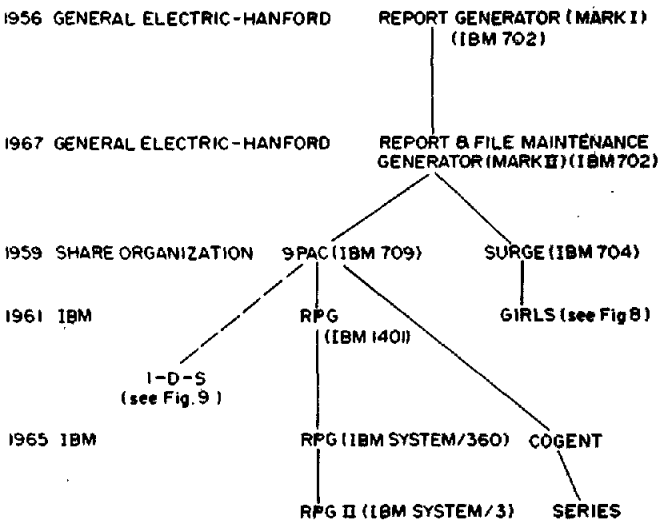

FIGURE 6. The Hanford/RPG Family.

ment of families during the period 1964–1968; and the vendor/CODASYL developments from 1968 to the present. Since the characteristics of the data-base management systems differ considerably during these periods, we discuss them separately.

Early Developments: Prior to 1964

The impetus for DBMS development came originally from users in government, particularly from the military and intelligence areas, rather than from industry and computer manufacturers. Although these prototypes bear little resemblance to today's systems and were somewhat isolated, they provided some interesting "firsts" in the evolution of data-base technology. They also provided the beginnings of several significant DBMS families.

In 1961 Green [X2] and his colleagues developed a natural-language system called BASE-BALL. Though not a data-base management system by current definition, it made a contribution to the technology by providing access to data through a subset of natural language (a limited vocabulary of baseball-related terms). At approximately the same time, the first implementation of a B-Tree was described by Collilla and Sams [X6].

Cheatham and Warshall were probably the first to discuss the translation of a query language. They designed a language, QUERY [X7], and developed techniques for analyzing its syntax and compiling statements into machine code.

One of the first identifiable data-base management systems to appear in the literature was an elegant generalized tape system developed by Climenson for the RCA 501 in 1962. This system, called Retrieval Command-Oriented Language [X8], provided five basic commands, with Boolean statements permitted within some of them. The user had to specify the data description with the query so that a program could be bound to its data.

Another early and ambitious development was ACSI-MATIC [X1] sponsored by the US Army in the late fifties. This system was designed by Minker to emphasize effective memory utilization and inferential processing. It could make inferences such as: if John is the son of Adam, and Mary is the sister of John, then Mary is the daughter of Adam. It contributed the first generalized data-retrieval accessing package for a disk-oriented system with batched requests, a dynamic storage algorithm for managing core storage, and the first assembler to use a dynamic storage allocation routine. Because disks were not reliable at that time, the ASCI-MATIC system was never fully imple-

mented. A prototype version was implemented later at RCA (1964).

The US Air Force also pioneered development of DBMS by sponsoring several projects at the MITRE Corporation. The prototype, called Experimental Transport Facility (ETF), led to the Advanced Data Management System (ADAM) [X24, 25, 27, 33], initiated in 1962. ADAM was implemented on an IBM 7030 (STRETCH) computer, with the design goals of providing a laboratory for modeling, prototype development, design verification, and evaluation of DBMS. Although ADAM did not meet all its ambitious design goals [X37] (many have not yet been achieved anywhere), it still remains a significant contribution to the technology.

The COLINGO system [X22], a contemporary of ADAM, was really a series of tape-oriented data-base systems with COBOL-like logical data structures implemented on the IBM 1401 computer system. The C-10 [X30] system, originally named as a follow-on to COLINGO, was implemented on the IBM 1410 computer and embodies many of the ADAM concepts. The influence of ADAM can be seen in System Development Corporation's LUCID [X13, 21], and in parts of Auerbach's DATA MANAGER-1 (DM-1) [X31].

Establishment of Families: 1964–1968

During this period the isolated developments diminished and full-scale families of DBMS emerged, some borrowing heavily from the past, others from sibling developments. A family is not limited to one company or government agency; because of the mobility of its developers, a family may spread across organizations, providing cross-fertilization of ideas. Although the family lineages of DBMS are sometimes intertwined, each can be traced to its progenitor.

The Postley/Mark IV Family (Figure 8)

One early system, which evolved into the MARK IV family, was GIRLS (Generalized Information Retrieval and Listing System), developed for the 7090 by Postley [X4]. Influenced by the SHARE SURGE development (as discussed on page 20), Gesc led successively to the development of the MARK I, MARK II, and MARK III systems for the IBM 1401/60 at Informatics between 1961 and 1967. In 1968 the highly successful MARK IV SYSTEM [V12] was released for use on the IBM System/360. Since then, numerous releases of MARK IV have provided over twenty new features, and MARK IV has now been implemented on other hardware.

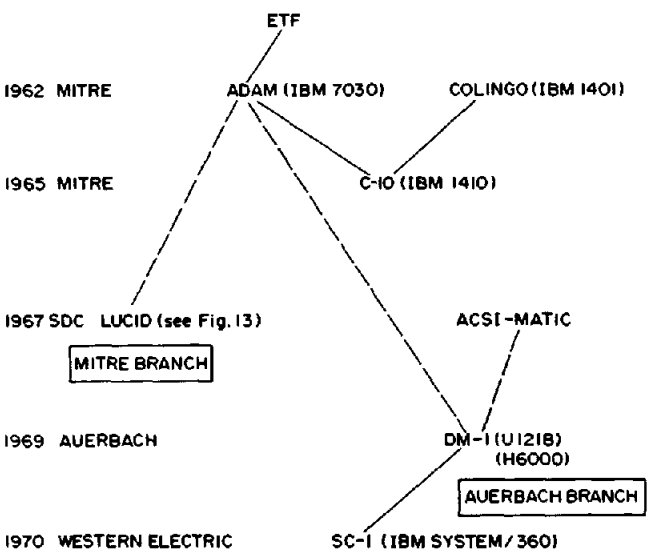

FIGURE 7. The MITRE/Auerbach Family.

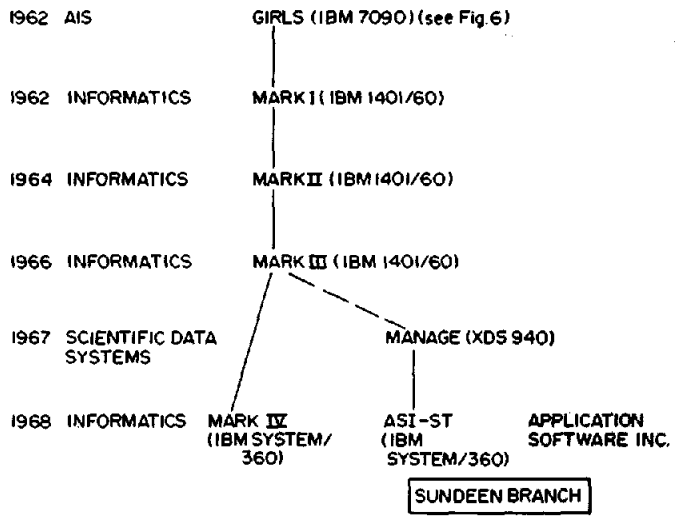

FIGURE 8. The Postley/MARK IV Family.

A significant offshoot of the Postley/MARK IV family is the Sundeen branch. This spans two different companies, starting with the MANAGE System [X23, Y4] developed at Scientific Data Systems, and followed by the AS-IST system [V1] developed at Applications Software in 1967 for the IBM System/360.

Bachman/IDS Family (Figure 9)

The Integrated Data Store (I-D-S) [X15, 18] was developed by Bachman and his colleagues at the General Electric Company in 1964. The I-D-S system, which stems from the same needs as 9 PAC, combined random-access storage technology with high-level procedural languages (GECOM in 1963, and COBOL in 1966) to provide a powerful network model of data. Significant I-D-S developments included:

- new data manipulation verbs or procedure calls at the high-level language interface;
- separate storage- and program-level item descriptions;

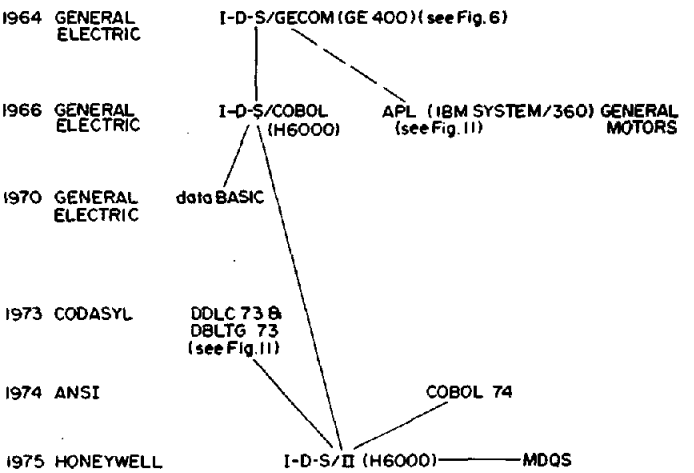

FIGURE 9. The Bachman/IDS Family.

- implicit insertion and removal of groups from relationships, based upon selection and ordering rules;
- retrieval of, and modification to both primary and secondary keys;
- data paging concepts based on logical data-base keys;
- incremental recovery and restart using "before" and "after" images; and
- shared access to the data base, with automatic detection of interference and automatic restart capability.

Since 1964 the I-D-S system has evolved under several different hardware systems, operating systems, and host languages. Recently, a new version, I-D-S/II [V9], using COBOL 74 [PL7], has been made available by Honeywell. It is consistent with the CODASYL DDLC 73 specification [S3] which will be discussed in the section on the CODASYL/ DDLC 73 specification [S3] which will be discussed in the section on the CODASYL/ DBTG Family, on page 25, and with recent COBOL additions.

In 1966 Dodd and his colleagues at General Motors Research developed APL (Associative PL/I) [X28], which is a development somewhat similar to that of I-D-S, but intended to provide data-management functions for a computer-aided design environment [G11]. APL provides six data-manipulation verbs in a PL/I host-language environment: CREATE, INSERT, FIND, FOR EACH, REMOVE, and DELETE. Another contribution of APL was the introduction of a distinct technology which separated logical relationships of the owner and member groups from their physical implementation.

Another branch in the I-D-S family is the dataBASIC system [V11]; implemented by Dressen at General Electric (now Honeywell) in 1970. This system offered the nonprogramming user high-level access to homogenous files (single record type) in a time-sharing environment using the BASIC programming language. Its only retrieval statement consists of the FOR (Boolean search statement), which qualifies a set of groups (records) to be retrieved. Each retrieval is processed by any number of processing statements until a concluding NEXT statement is encountered.

A recent offshoot in the I-D-S family is the Honeywell Management Data Query System, MDQS [V10]. This system is a self-contained query and report specification facility to access sequential, index sequential, and I-D-S files.

Formatted File/GIS Family (Figure 10)

At about the same time as the host language progenitor (9PAC) was evolving, a series of government systems was being developed to

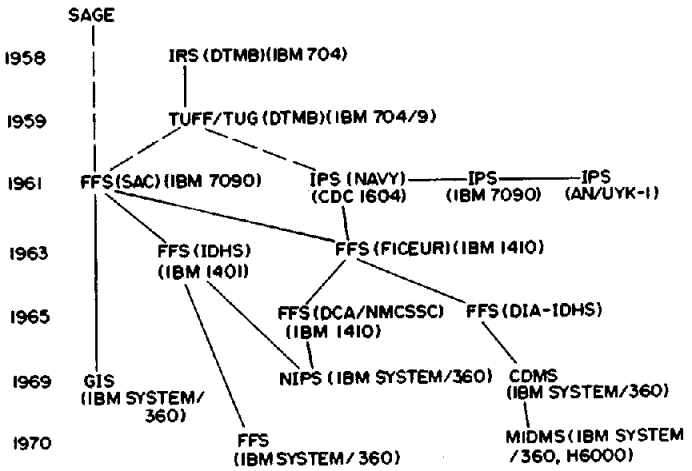

FIGURE 10. The Formatted File/GIS Family.

support the needs of the Command-and-Control and the Intelligence communities. Perhaps the most prolific of these was the Formatted File family, which spans all three development periods. Its origins can be traced to a series of systems developed at the David Taylor Model Basin[3] by Davis, Todd, and Vesper. One of the principal systems—Information Retrieval (IR) [X3, 9]—was an experimental prototype developed in 1958 for the IBM 704. This was followed by two formatted file-processing packages: Tape Update for Formatted Files, TUFF [X16, 20], and Tape Updater and Generator, TUG [X5] (both developed to run on the IBM 704). Later this family split into two branches in the Air Force and Navy. The Air Force branch, SAC/AIDS Formatted File System [X14], was developed in 1961 for the Strategic Air Command 438L system. Its major contribution to data technology was the development of a file format table, i.e., a "self describing" data base. By storing a machine-readable data definition with the data, each data base was directly accessible by FFS.

The Navy branch, Information Processing System (IPS) [X11, 12, and Y10], was also developed in 1963 for the CDC 1604 by NAVCOSSACT. IPS also made contributions to data-base technology in the implementation of a multilevel hierarchically structured data base on sequential media, and in its implementation on several different hardware systems, such as the IBM 709/90 [X19] and the AN/FYK-1 [X32].

During the implementation of IPS in 1963, another branch of the family was developed for the Naval Fleet Intelligence Center in Europe (FICEUR) [X10]. This FFS was patterned after the SAC FFS and implemented on the IBM 1410. SAC also added an FFS on the IBM 1401 for the Pacific Air Force Headquarters. This system was later reprogrammed for the IBM System/360 and is still in use on smaller models.

About 1965 the SAC and FICEUR branches of the formatted-file family merged, resulting in the NMCS Information Processing System (NIPS) [X17]. NIPS added the concepts of logical file maintenance, improved query language, and on-line processing. In 1968 NIPS was converted from IBM 1410 to IBM System/360 and named NIPS-360 [Y12].

A cousin of NIPS was also developed for the intelligence community—the Intelligence Data-Handling Formatted File System [X26]. This emphasized efficient large-file processing and provoked interest in machine-independent implementation using COBOL. Prototype development of such a system began in 1968 by the Defense Intelligence Agency. The effort was first named the COBOL Data Management System (CDMS) [Y8]; later (1970) it was renamed the Machine Independent Data Management System (MIDMS) [Y11]. It was originally implemented on the IBM System/360 and was later coded (in 1973) for the H6000 series.

SAC FFS is considered to have inspired IBM's Generalized Information System (GIS) [V16, 17]. This was originally developed as a stand-alone program product for System/360 (1965), but has been extended and enhanced to act as either a stand-alone system or ad hoc interrogation interface for the IMS family.

Vendor/CODASYL Developments: 1968 to the Present

The trend in this period shifts from in-house family-oriented activities to proprietary vendor development. As a result, some advances made by commercially available DBMSs disappeared into a veil of secrecy. While few references have appeared recently on the internals of particular DBMSs, the technical literature abounds with articles on mathematical and theoretical aspects, especially of relational systems. Chamberlin's article (see page 43) provides an excellent bibliography of this development. Recent years also show the entry of CODASYL into the data-base field.

CODASYL/DBTG Family (Figure 11)

Based upon the pioneering ideas of I-D-S and APL, the CODASYL Programming

[3] The David Taylor Model Basin is now called the David Taylor Naval Ship Research and Development Center.

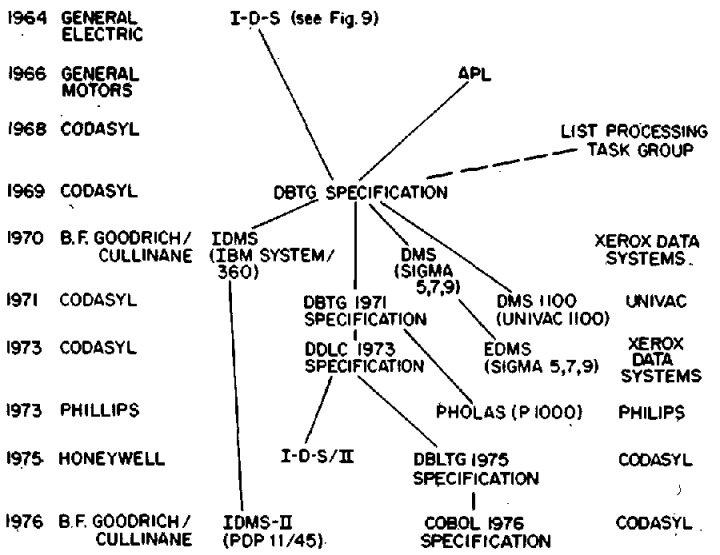

FIGURE 11. The CODASYL Family.

Language Committee started a new task group to work on a proposal for extending COBOL to handle data bases [PL6]. This group was originally called the List Processing Task Group, though its name was later changed to the Data Base Task Group—DBTG—its major acronym, which will be used here. The first semipublic recommendations of the DBTG were made in 1969 [S1]. These recommendations detailed the syntax and semantics of a Data Description Language (DDL) for describing network-structured data bases, and the definition of Data Manipulation Language (DML) statements to augment COBOL. The task group intended that the DDL specifications should be available to all programming languages, while extensions like the DML would be needed for every language.

The initial DBTG specification was reviewed by many user and implementation groups. Their recommendations were further considered, and a new report was issued in 1971 [S2]. The major change involved separation of the data description into two parts; a Schema DDL for defining the total data base, and a Sub-schema facility for defining various views of the data base consistent with different programming languages.

Based on the reviews of the 1971 report, CODASYL took two significant actions:

- a new standing committee was created to deal exclusively with the data description, the Data Description Language Committee (DDLC); and
- the DBTG was replaced by a new task group to deal only with COBOL extensions, the Data Base Language Task Group (DBLTG).

Since that time, a new subcommittee has also been formed to add DML statements to FORTRAN.

The DDLC was charged with taking the Schema DDL and developing a common data description language to serve the major programming languages. In January 1974 a first issue of the Data Description Language Committee's publication, the Journal of Development, was published [S3]. This report specifies only the syntax and semantics of the DDL.

The DBLTG was charged with making the 1971 report of the DBTG consistent with CODASYL COBOL specifications. In February 1973 the DBLTG submitted its report to the CODASYL Programming Language Committee. This report is very similar to the 1971 DBTG report, with nomenclature and relatively cosmetic changes. New items in the 1973 report included an ex-

tension to the facility for dealing with error returns.

Implementation of systems which conformed to the 1969, 1971, and 1973 DBTG specifications started in 1970 with the UNIVAC DMS 1100 [V22] for the 1108, and since then for the UNIVAC 1110 series computers. At about the same time, B. F. Goodrich implemented a system called Integrated Data Management System, IDMS [V7], for the IBM System/360. This has since been extended to IDMS-11 for the Digital Equipment Corporation PDP 11/45. The IDMS series is marketed by Cullinane Corporation. The Digital Equipment Corporation has implemented DBMS-10 [V8] for its PDP 10 computer system.

Some extensions to self-contained facilities for ad hoc interrogations have been implemented by Control Data Corporation, Query/Update [V6], and by Xerox Data Systems, EDMS [V23]. In the Netherlands, Philips implemented a family of systems termed PHOLAS [V19], and in Norway the SIBAS [V20] system has been developed by Shipping Research Services. Honeywell has updated I-D-S to conform to 1973 specifications; this is the I-D-S/II [V9].

IMS Family (Figure 12)

The IMS family of systems is an outgrowth of the Apollo moon-landing program. Its origins can be traced to two developments at The Space Division of North American Aviation (now Rockwell International) in 1965. One was the implementation of GUAM, (Generalized Update Access Method), the forerunner of Data Language/One (DL/I). The other was the implementation of two teleprocessing applications, EDICT (Engineering Document Information Collection Task) and LIMS (Logistics Inventory Management System). The software package which supported EDICT and LIMS, the Remote-Access Terminal System (RATS), was jointly developed by Rockwell International and IBM during 1964–65. Both GUAM and RATS were originally implemented on the IBM 7010 with 1301 disk storage.

In 1966, IBM, Caterpillar Tractor Corporation, and Rockwell International agreed to a joint development effort to produce a DBMS, the Information Management System (IMS) for the IBM System/360. When the system had to be frozen in 1968 (to meet the Apollo commitment), Rockwell and IBM each continued with separate developments, while Caterpillar withdrew entirely from the effort. The development at Rockwell took the name of Information Control System/Data Language/I (ICS/DL/I).

Originally, DL/I [X35] was a data description facility which provided a means for describing and organizing a hierarchically structured data base. It also provided interfaces, which the programming user invoked to access and store data from the host language (originally COBOL). The on-line component, ICS/DL/I [X34], added in 1968, allowed multiple access by using the DL/I interface from COBOL or PL/I programs. In addition to running teleprocessing simul-

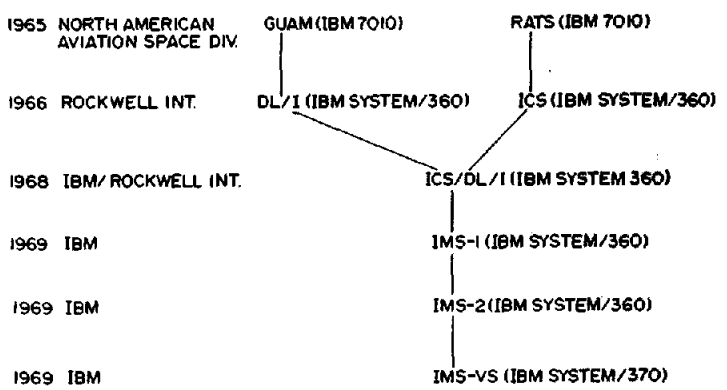

FIGURE 12. The IMS Family.

taneously with batch processing, the system handled several remote terminals.

In 1969 IBM released its version, the Information Management System/360 (IMS/360) [V13]. Since 1969 a series of improvements has marked its evolution [V14, 15].

Inverted File Family (Figure 13)

Following the LUCID System development in 1968, the Advanced Research Project Agency (ARPA) sponsored System Development Corporation's development of the Time-Shared Data Management System, TDMS [Y1, 2, 3, and X29]. This was designed to operate in the time-sharing environment of the ADEPT executive on the IBM System/360. It was the first DBMS to combine an inverted file implementation of hierarchical data model with interactive processing.

In 1966 the Computation Center at the University of Texas began the development of a Remote File Management System (RFMS) [Y7] on its CDC 6000. RFMS differed from TDMS mainly in its internal design. A version of RFMS was marketed by CDC as MARS VI [V5].

MRI Systems Corporation (whose principals were originally associated with the University of Texas) continued development of an RFMS under the name of SYSTEM 2000 [V18], which was offered commercially in 1970. A number of significant enhancements have been made since 1970 so that SYSTEM 2000 offers an integrated set of host-language and self-contained capabilities.

Additional Vendor Developments

A variety of other data-base management systems based on inverted files for efficient query processing were developed during this period by other vendors. Two of the more commonly known are ADABAS [V21], developed by Schoell at Software AG (West Germany), and Model 204, developed by Computer Corporation of America [V4].

ADABAS uses the inversion tables not only for efficient retrieval, but also for linkages between records of different files. ADABAS provides access to the data through a host language interface, a self-contained language for on-line inquiry, and a batch report generator. ADABAS is one of the few systems which offer a data compression facility.

The Model 204 query language provides most of the power of a general-purpose programming language from an on-line terminal, but is easy to use for simple requests. This system uses the IFAM access method to allow multiple field indexing and variable length records for file compression as well as for text processing.

Three other vendor developments date back to about 1969, TOTAL, DM-1, and DMS II. Although in its initial release, TOTAL [V3] was primarily a direct access data-base management system, facilities were soon added to process DBTG-like sets implemented with chain pointers. TOTAL is a host-language system, which can model the major data structures of the DBTG specifications, and it was one of the first systems to offer a Schema-Sub-schema processor fa-

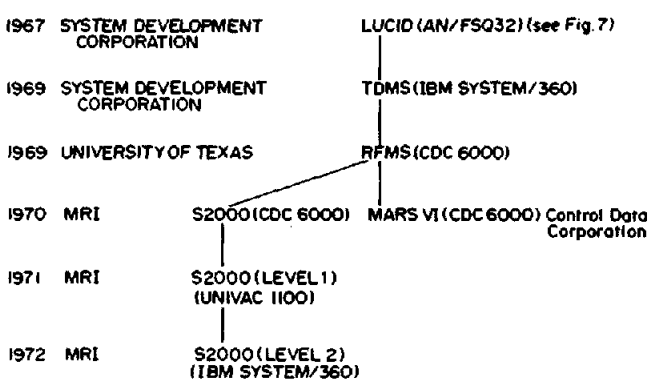

FIGURE 13. The Inverted File Family.

cility. It has become one of the most widely used data-base management packages today.

The Data Manager-1 System (DM-1) [X31], designed by Sable at the Auerbach Corporation, stems from the Army ACSI-MATIC development and MITRE'S ADAM. DM-1 consists of a series of service routines for returning and storing data; using these routines, both high-level ad hoc user functions and host-language application programs can be developed. DM-1 was implemented at the Air Force Rome Air Development Center on U1218 computer and the Honeywell H6000. Based on the design philosophy of DM-1, the Western Electric Company, initially assisted by Auerbach, developed System Control-1 [Y6] on the System/360.

Another development, by the Burroughs Corporation, is the Data Management System II [V2] for the B6700/B7700 computer. Basically a host-language type system using COBOL, its data definition language is formed in set-theoretic terms. It also offers a storage definition option.

6. THE PRESIDENTIAL DATA BASE EXAMPLE

The discussion of data-base models in other articles in this issue of COMPUTING SURVEYS will use a unified example which deals with some parts of the Executive branch of the US Government, with data about the President, his Administration, elections, Congress, etc. We use this example because it is almost self-explanatory; it was first enunciated in a paper by Willner, et al. [G9].

Because the example deals with the Executive branch, the most obvious entity is the PRESIDENT. The important items in the PRESIDENT entity will be assumed to be: the President's name (PRES-NAME), BIRTH-DATE and DEATH-DATE, the party affiliation (PRES-PARTY), and the name of his SPOUSE. It will also be considered necessary to know the STATE-NAME of which the President is a native son. However, since STATE will later be defined as an entity, we could alternatively define a relationship NATIVE-SON between PRESIDENT and STATE.

Using the notation presented in Section 3 under the discussion of the "Elements of Logical Structure" (page 13) we have Display 1 below. If, however, an explicit relationship were to be used for the native son, and STATE-NAME is the key of STATE then the statement appears as in Display 2 below.

The next entity of interest is the President's ADMINISTRATION, which contains items such as the administration number (ADMIN-NUMBER) (e.g., George Washington was No. 1), the inauguration date (INAUG-DATE), and the Vice-President (VP). In order to identify the President of each Administration, it is also necessary to include the item PRES-NAME in the ADMINISTRATION entity.

At this point, it is worth asking why the PRESIDENT entity does not *contain* the ADMINISTRATION entity. This is a design decision, and the reader must assume it is based on consideration of usage and modeling. It should be noted, however, that a President can have had more than one Administration, and consequently, if ADMINISTRATION is contained, it would need to be a *repeating group*. As another alternative, we could assume that the two separate entities have a relationship HEADED between ADMINISTRATION and PRESIDENT. Thus, we have Display 3) below.

Display 1:
PRESIDENT = ⟨PRES-NAME, BIRTH-DATE, DEATH-DATE, PRES-PARTY, SPOUSE, STATE-NAME⟩

Display 2:
PRESIDENT-1 = ⟨PRES-NAME, BIRTH-DATE, DEATH-DATE, PRES-PARTY, SPOUSE⟩
and
NATIVE-SON = ⟨PRES-NAME, STATE-NAME⟩.

Display 3:
either
⟨ADMINISTRATION⟩ = ⟨ADMIN-NUMBER, PRES-NAME, INAUG-DATE, VP⟩;
or:
PRESIDENT-2 = ⟨PRES-NAME, BIRTH-DATE, DEATH-DATE, PRES-PARTY, SPOUSE, STATE-NAME, {⟨ADMIN-NUMBER, INAUG-DATE, VP⟩}⟩;
or:
ADMINISTRATION-1 = ⟨ADMIN-NUMBER, INAUG-DATE, VP⟩
HEADED = ⟨PRES-NAME, ADMIN-NUMBER⟩.

The next entity is that of the ELECTION. The interesting items in the election are: the year (ELECTION-YEAR), the presidential votes in the Electoral College (PRES-VOTES), the LOSER, the LOSER-PARTY, the year in which the party was first created as a political entity (PARTY-FIRST-YEAR), and the votes of the losing party (LOSER-VOTES). Once again, because elections are won by a President, the election entity may have to contain the PRES-NAME; otherwise there must be some relationship WON between the PRESIDENT and the ELECTION entities. Thus, the alternatives are:

ELECTION = ⟨ELECTION-YEAR, PRES-NAME, PRES-VOTES, LOSER, LOSER-PARTY, PARTY-FIRST-YEAR, LOSER-VOTES⟩, etc.

Another entity within the data base is the STATE. It has a name (STATE-NAME), a population (POP), and a number of votes in the Electoral College (STATE-VOTES). States are admitted to the Union during some Administration. This fact may be shown either implicitly, by having some relationship (ADMITTED-DURING) between the ADMINISTRATION and STATE entities, or explicitly, by including the ADMIN-NUMBER in the STATE entity. It might be noted that there is already a link between the PRESIDENT and STATE entities because the NATIVE-SON relation has been shown as an element (STATE-NAME) in the PRESIDENT entity.

We have now defined most of the data base, and need only incorporate the entity CONGRESS to complete it. This entry will contain items such as: CONGRESS-NUMBER, SENATE-REPUBLICAN-PERCENT, SENATE-DEMOCRAT-PERCENT, HOUSE-REPUBLICAN-PERCENT, AND HOUSE-DEMOCRAT-PERCENT. Again, there is a relation between the PRESIDENT and CONGRESS, which may be found explicitly by incorporating PRES-NAME in the CONGRESS entity, or implicitly by arranging a relation CONGRESS-SERVED between the entities.

Figure 14 shows a sample of the presidential data base in tabular form. Unavailable information is shown by a ϕ, e.g., in the Death and Inauguration Date columns.

But there are some drawbacks to this example: one is the fact that it represents a relatively constant data base, for although a President may be replaced, the data about the Administration is still retained. Consequently there is little *updating* in our example, though there may be substantial *addition* to the data base in election years. Some business data bases, however, present a greater propensity to change. For example, a payroll data base regularly has changes to many items such as YEAR-TO-DATE-PAY (presumably after every payday) and SALARY (presumably after every increase). Thus, the presidential data base, while forming the major example, will not suffice alone. Other authors contributing to this issue of COMPUTING SURVEYS will introduce other examples to illustrate particular fine points.

7. TRENDS AND ISSUES

Historically, we have traced the development of DBMS from the early systems, which supported primarily the nonprogramming user for ad hoc requests, to the recent predominance of host-language systems which support the programming user. A current trend is, then, the establishment of a balance—a comprehensive set of DBMS functions for a full spectrum of users while maintaining the current DBMS objectives [F1, 2, and 3]. Some of the current research is developing bridges between various models of data so that a single DBMS can support a variety of data models.

Three major trends and one important issue will affect the future of DBMS: the emergence of conversational systems, the need for geographic distribution of the information system, the technological impacts on DBMS architecture, and the question of standardization of the DBMS interface. Each of these is now briefly discussed.

Ad Hoc versus Programming Systems

Artificial intelligence research has already improved our understanding of the difficulties involved in providing a natural language interface for computers. And though there

has been little that is immediately applicable, the fall-out from this research includes a better understanding of the structure and use of higher-level and very-high-level (restricted natural) language interfaces. As a result, some DBMS already provide good languages for the nonprogrammer who is willing to learn a few rules, and there is growing interest in the development of the casual-user interface (e.g., see [F4]).

PRESIDENT

PRES-NAME	BIRTH-DATE	DEATH-DATE	PRES-PARTY	SPOUSE	STATE-NAME
Eisenhower	10/14/1890	03/28/1969	Republican	Mamie	Texas
Kennedy	05/29/1917	11/22/1963	Democrat	Jacqueline	Mass.
Johnson	08/27/1908	01/22/1973	Democrat	Claudia	Texas
Nixon	01/09/1913	ϕ	Republican	Patricia	Calif.
Ford	07/14/1913	ϕ	Republican	Elizabeth	Mich.

ELECTION

ELECTION-YEAR	PRES-NAME	PRES-VOTES	LOSER	LOSER-PARTY	PARTY-FIRST YEAR	LOSER-VOTES
1952	Eisenhower	442	Stevenson	Democrat	1824	89
1956	Eisenhower	457	Stevenson	Democrat	1824	73
1960	Kennedy	303	Nixon	Republican	1856	219
1964	Johnson	486	Goldwater	Republican	1856	52
1968	Nixon	301	Humphrey	Democrat	1824	191
			Wallace	3rd Party	1968	46
1972	Nixon	520	McGovern	Democrat	1824	17

CONGRESS

CONGRESS-NUMBER	PRES-NAME	SENATE-REPUB-LICAN-PERCENT	SENATE DEMOCRAT-PERCENT	HOUSE-REPUB-LICAN-PERCENT	HOUSE-DEMOCRAT-PERCENT
83	Eisenhower	50%	49%	49%	50%
84	Eisenhower	49%	51%	47%	53%
85	Eisenhower	49%	51%	46%	54%
86	Eisenhower	34%	66%	35%	65%
87	Kennedy	36%	64%	40%	60%
88	Kennedy Johnson	33%	67%	41%	59%
89	Johnson	32%	68%	33%	67%
90	Johnson	36%	64%	43%	57%
91	Nixon	43%	57%	44%	56%
92	Nixon	44%	54%	41%	59%
93	Nixon Ford	42%	56%	44%	56%
94	Ford	37%	60%	33%	66%

Figure 14. A sample of the presidential data base.

STATE

STATE-NAME	ADMIN-NUMBER	POP	STATE-VOTES
Texas	16	11196730	26
Mass.	ϕ	5689170	14
Calif.	18	19953134	45
Mich.	12	8875083	19

ADMINISTRATION

ADMIN-NUMBER	PRES-NAME	INAUG-DATE	VP
1	Washington	04/30/1789	Adams
2	Washington	03/04/1793	Adams
16	Polk	03/04/1845	Dallas
18	Fillmore	07/10/1850	ϕ
49	Eisenhower	01/20/1953	Nixon
50	Eisenhower	01/20/1957	Nixon
51	Kennedy	01/20/1961	Johnson
52	Johnson	11/22/1963	ϕ
53	Johnson	01/20/1965	Humphrey
54	Nixon	01/20/1969	Agnew
55	Nixon	01/20/1973	Agnew Ford
56	Ford	08/09/74	Rockefeller

Fig. 14. (contd.): A sample of the presidential data base.

A *casual user* is one who uses the system so seldom that all rules and techniques are likely to be forgotten between sessions, hence the need for special treatment. At the other end of the user spectrum are the adept computer programmers who have technical skills and a good knowledge of "system internals." In writing programs for nonprogrammers they presumably utilize all their skills to produce procedures that will run efficiently. The assumption is that programmers cost more (they must be paid while they understand the problem, write code, etc.), but their resulting programs are cheaper to run.

Thus, the case for ad hoc and host-language systems can be considered one of tradeoffs. The following is a partial list of the advantages and disadvantages of the use of higher-level interfaces:

1) Their use facilitates more rapid running of the problem—the user asks the question directly, and he has no need to call on a programmer as intermediary (a process that sometimes takes weeks for even a simple problem in a busy industrial environment). This advantage is offset by the relatively high cost of using what is essentially an interpretive system: the tradeoff is therefore between people and machine costs. The people costs are in programming and debugging, while the machine costs are in running. One presumes that the code produced from a high-level (query) interface costs more to run, therefore the question arises: how many times must the program be run before it pays for the cost of programming? And this is the classical question of compiling, but now in the realm of even higher-level languages and with potentially large data bases. There are, however, very few jobs today which warrant the cost of special (assembler or machine language) programming. This trend continues today in DBMS usage, and the self-contained ad hoc user system is becoming more accepted by the user community.

2) The use of a higher-level language simplifies the structure (removes DO-loops and GO-TO statements) and is generally

more understandable, consequently less error prone. On the other hand, a simple question may invoke a long and costly procedure; e.g., "Give me the average height of all Americans," may involve a sequential search of 200 million records! Also, the possibility for ambiguity immediately arises. The request "Give me the count of all people in New York," could be interpreted as ". . .all people who are, at this instant, in the state of New York," while the questioner intended to ask ". . . all people who have, as their residence, the city of New York." The trouble with this question is obvious, but the user may never realize that the answer given was not correct for the intended question.

3) The very-high-level languages tend to have a mathematical equivalence—they can be transformed into precise mathematical formulas (e.g., in predicate calculus). They are therefore capable of exact checking. In this way, the potentially ambiguous statement can be transformed into an exact statement and "played back" to the questioner, thereby helping to eliminate error. The high-level program, however, does not have an exact statement of its operation in good mathematical terms; it does what the programmer told it to do, good or bad (and all too often the latter). Precision of statement is an advantage to the mathematically sophisticated user, and possibly to others as well.

Thus, the user trend may well be toward the higher-level-language interface, but for years to come it will be necessary to program the large and repetitive systems of industry and government efficiently by using the language interfaces currently in use (e.g., COBOL, FORTRAN, PL/I).

Geographically Distributed Systems

Inexpensive communication between computing systems, and the development of national and international networks have forced further changes on the design of computing systems. In this, DBMS is no exception. The concept of distributed data bases, where a processor calls on data at several other locations, is already a reality on some homogeneous systems—and possibly (with difficulty) on some nonhomogeneous systems. This trend may be seen, in part, as an answer to the wish of industry and government to access its data in reasonable time.

As an example, one major corporation found that the use of a computer network allowed it to strike a corporate dollar balance each Friday; thus, the company could let the money out on short-term loan (over the week-end). Surprisingly, the money realized as interest on the loan paid for all the network facilities. Similarly, in many large corporations, the warehousing cost is great; all material resting in inventory represents an unprofitable capital expenditure. Large retail merchandising companies can reduce inventory costs by knowing *what* is available *where* in their many warehouses, and thus be able to reduce surplus stock. Some large corporations have been able to give their sales forces remote access to their computer systems, thereby allowing the salesman (and through him, the customer) direct on-line access to shipping and pricing information. The competitive advantage is very high in such cases.

Distributed systems, then, show a need for:

1) computers to be networked. It is not generally possible to have all the power at a central site, and each major node (e.g., the largest warehouses) has its processor.
2) data to be distributed. If the data is entered at hundreds of locations throughout the country, it is probably efficient to store it near the entry port. In some large banking systems, the customer accounts are kept in the computer system at the local banks, but other branches can still service the customer (and debit the account!).

But distributed systems pose many new problems, and exacerbate many old ones. Some of the new problems are revealed in the following questions.

- How do we change the request language? Does the user have to know the location of the data? Is there a central data dictionary/directory? Can the user request data by broadcasting a message for it?

- Is it better to store multiple copies? How much extra will it cost to update a data base from a remote location? What parts of the data base should be stored (i.e., how does one distribute the data efficiently)? What are the best places to run a program (it may be cheaper for a user at A to transport data at B to the program at C and then just receive the answers at A)?

The old problems have already been discussed, but are now complicated by the extra complexity of the distributed system:

- What redundancy is necessary to ensure good reliability of both hardware and data? How much does this affect the user in terms of the response time for updates, and the excess processing cost?
- What problems are likely to occur in concurrent operation? The possibility that several users will all contend for the same resources, and consequently will need effective scheduling and control, is obviously more acute in a large, distributed, many-user system.
- How can privacy be retained? The potential for breaking the system rises as its complexity increases. The chance of message interception obviously increases also.

Thus, the trend to distributed data bases, with concepts of data machines as special resource nodes on the network, brings with it a new set of tradeoff decisions.

Data-Base Machines

Distributed data bases, in conjunction with emerging technology, will have a significant impact on DBMS architecture and on the DBMS functions. There already are computers dedicated to DBMS, e.g., the Datacomputer [F5, 6]. "Front-end" and "back-end" computers are in the prototype stage [F7, 8]. Also, new disk technologies and associative devices will have a great impact on DBMS architecture [F9, 10].

To Standardize or Not?

The computing profession has ambivalent feelings about standardization: everyone seems to admit it has merits, but finds excuses in order to stop it from happening too soon in his own field of interest. The arguments for and against standardization (in any area) are now given.

For standards, there is one major argument: The provision of a standard aids the user by making objects interchangeable; the nut, if of the same diameter, fits the bolt. Thus:

- the programming language is the same on all machines: so the programmer who knows COBOL, for example, can be transferred, or may get a new job and not need retraining;
- the company can change machines and run the same COBOL programs, after their recompilation, on the new machine;
- parts are interchangeable: magnetic tapes have standard densities; plug-to-plug compatibility of storage and input/output units is possible;
- data can be interchanged over the network;
- the network protocol is the same, so all users have to learn only one protocol; and
- the commands to enter (log-on) and leave (log-off) the system, and some other controls, are the same throughout the network.

Against standards, there is one major argument: if we do not know the correct technology, standardization may mean costly refitting later, or may even stifle development. This argument is reasonable, since a large-scale data-processing shop may have many thousands of programs representing millions of dollars of investment. Rewriting all these (probably COBOL) programs in some new language is beyond the wishes of most current DP managers, who hope that their programs are "here to stay." Such built-in conservatism will undoubtedly slow down any change from one well-developed standard to another, no matter how good the new standard may be. This stifles acceptance of new ideas.

Many groups are concerned about standardization and are actively working in this area. The DBTG report has been accepted by the Programming Language Committee

of CODASYL as a part of JOD COBOL. The ANSI/X3/SPARC/Study Group on Data Base Systems has been meeting since 1972. Part of their charge is to develop a basis for DBMS standardization. Their recent report [F11] formulates many functional interfaces of a DBMS. The languages used to communicate across these interfaces may be candidates for standardization.

There are therefore many potential areas for standardization of DBMS:

- the definition language for the logical structure;
- the language(s) to manipulate the data;
- the protocols for invoking procedures on the data-base machine;
- the protocols on the network of a distributed system; and
- the storage devices and physical mapping of data.

Each of these has its proponents and opponents for various kinds of system models. As a result, the issue of standardization is a mixture of common sense, politics, economics, philosophy, convenience, and taste. Many researchers consider standards an anathema, but many users see standards as a necessity. The arguments will still be going on fifty years from now (even though there will undoubtedly be DBMS standards by then).

ACKNOWLEDGMENT

The Guest Editor's Introduction to this issue of COMPUTING SURVEYS has already expressed gratitude to the wide variety of experts who made this article possible. This includes developers of DBMS who implemented the systems and who helped us correct errors in the history, as well as our reviewers. V. Kevin Whitney and Richard G. Canning in particular made many valuable suggestions.

CLASSIFICATION OF REFERENCES

A Data Administration
D Data Dictionary
DL Data Definition Language
F Future, Trends
G General
I Introductory
M Data Models—Theory
PL Programming Languages
S DBMS Specifications
SL Stored-Data Definition
T DBMS Texts
U Surveys
V Vendor Systems
W Report Generator
X DBMS Prior to 1968
Y DBMS 1968 to Present
Z Relational Systems

REFERENCES

(A) Data Administration

[A1] EVEREST, G. C., "Data base administrator organizational role and functions," MISRC-WP-73-05.

[A2] GUIDE INTERNATIONAL, "The data base administrator," Nov. 1972.

[A3] CANNING, R. G. (Ed.), "The data administrator function," *EDP Analyzer* **10**, 11 (Nov. 1972).

[A4] NOLAN, R., "Computer data bases: The future is now," *Harvard Business Review* (Sept. 1973).

(D) Data Dictionary

[D1] UHROWCZIK, P. P., "Data dictionary/directories," *IBM Systems J.*, **12**, 4 (Dec. 1973).

[D2] CANNING, R. G. (Ed.), "The data dictionary/directory function," *EDP Analyzer*, **12**, 10 (Nov. 1974).

(DL) Data Definition Language

[DL1] WILWORTH, N. E., "System data control," Tech. Memo. TM222/013/00, System Development Corp., Santa Monica, Calif., August 1975.

[DL2] D'IMPERIO, M., "Data structures and their representation in storage," in *Annual Review in Automatic Programming*, M. Halpern and C. J. Shaw (Eds.), Pergamon Press, Elmsford, New York, 1969, pp. 1–75.

[DL3] SENKO, M.; ALTMAN, E.; ASTRAHAN, M.; AND FEHDER, P., "Data structures and accessing in data-base systems," *IBM Systems J.*, **12**, 1 (1973), 30–93.

[DL4] SENKO, M. E., "Data description language in the context of a multilevel structured description: DIAM II with FORAL," IBM Research Report #6769, 1974.

[DL5] SENKO, M. E., "The DDL in the context of a multilevel structured description DIAM II with FORAL," in *Data Base Description*, B. C. M. Douque and G. M. Nijssen (Eds.), North-Holland Publ. Co., Amsterdam, The Netherlands, 1975.

(F) Future, Trends

[F1] WHITNEY, V. KEVIN, "Fourth generation data management systems," *Proc. of AFIPS National Computer Conf.*, 1973, Vol. 42, AFIPS Press, Montvale, N.J. 1973, pp. 239–244.

F2] BACHMAN, CHARLES, "Trends in data-base Management," *Proc. of AFIPS National Computer Conf.*. 1975. Vol. 44.

AFIPS Press, Montvale, N.J., 1975, pp. 569–576.

[F3] EVEREST, GORDON C., "The futures of data-base management," *Proc. 1974 SIGMOD Conf.*, May 1974, pp. 445–462.

[F4] CODD, E. F., "Seven steps to rendezvous with the casual user," *Proc. IFIP TC-2 Working Conf. on Data Base Management System Congress*, April 1974, North-Holland Publ. Co., Amsterdam, The Netherlands, 1974.

[F5] MARILL, THOMAS; AND STERN, DALE, "The datacomputer—a network data utility," *Proc. of AFIPS National Computer Conf.*, 1975, Vol. 44, AFIPS Press, Montvale, N.J., 1975, pp. 389–395.

[F6] MARILL, T.; AND STERN, D. DATACOMPUTER VERSION I USER MANUAL, Working paper no. 11, Computer Corp. of America, Cambridge, Mass., August 1975.

[F7] CANADAY, R. H.; HARRISON, R. D.; IVIE, E. L.; RYDER, J. L.; AND WEHR, L. A., "A back-end computer for data base management," *Comm. ACM* **12**, 10 (Oct. 1974), 575–582.

[F8] HEACOX, H. C.; COSLOY, E. S.; AND COHEN, J. B., "An experiment in dedicated data management," in *Proc. of Internatl. Conf. on Very Large Data Bases*, Sept. 1975, ACM, New York, 1975, pp. 511–513.

[F9] SU, S. Y. W.; COPELAND, G. P.; AND LIPOVSKI, G. J., "Retrieval operations and data representations in a context-addressed disk system," *Proc. ACM–SIGPLAN–SIGIR Interface Meeting on Programming Languages and Information Retrieval*, Nov. 1973, pp. 144–160.

[F10] LIN, C. S., AND SMITH, D. C. P., "The design of a rotating associative array memory for a relational data-base management application," *ACM TODS*, **1**, 1 (March 1976), 53–65.

[F11] ANSI/X3/SPARC/STUDY GROUP—DATA BASE SYSTEMS, "Interim report," *ACM/SIGMOD Newsletter:fdt*, **7**, 2 (Dec. 1975).

(G) General

[G1] MCGEE, W. C., "Generalization: key to successful electronic data processing," *J. ACM* **6**, 1 (Jan. 1959), 1–23.

[G2] MCGEE, R. C.; AND TELLIER, H., "A re-evaluation of generalization," *Datamation*, (July-August 1960), 25–38.

[G3] YAO, S. B.; AND MERTEN, A. G., "Selection of file organization using an analytic model," *Proc. of the Internatl. Conf. on Very Large Data Bases*, Sept. 1975, ACM, New York, 1975, pp. 255–267.

[G4] STEEL, T., "Beginnings of a theory of information handling," *Comm. ACM* **7**, 2, (Feb. 1964), 87–103.

[G5] ROSEN, SAUL, "Programming systems and languages—a historical survey," *Proc. of the Spring Jt. Computer Conf.*, 1964, Vol 25, AFIPS Press, Montvale, N.J., 1964, pp. 1–25.

[G6] ROSIN, ROBERT F., "Supervisory and monitor systems," *Computing Surveys* **1**, 1 (March 1969), 37–54.

[G7] DENNING, PETER J., "Third generation computer systems," *Computing Surveys* **3**, 4 (Dec. 1971), 175–216.

[G8] EVEREST, GORDON C.; AND SIBLEY, EDGAR H., "A critique of the GUIDE-SHARE data-base management system requirements," *Proc. of the 1971 ACM–SIGFIDET Annual Workshop on "Data Description, Access and Control,"* E. F. Codd and A. L. Dean, (Eds.), pp. 93–112, also MISRC-WP-71-2.

[G9] WILLNER, S. E.; BANDURSKI, A. E.; GORHAN, W. C.; AND WALLACE, M. A., "COMRADE data management system," *Proc. AFIPS National Computer Conf.*, 1973, Vol. 42, AFIPS Press, Montvale, N.J., 1973, pp. 339–345.

[G10] BACHMAN, C. W., "The programmer as navigator," *Comm. ACM* **16**, 11 (Nov. 1973), 653–658.

[G11] GARTH, W., "Design console technology at General Motors," *Proc. SHARE 1974 Conf.*, August 1974.

[G12] EVEREST, GORDON C., "The objectives of data-base management," *Information Systems COINS IV* (Tou), Plenum Press, New York, 1974, pp. 1–35, also MISRC-WP-71-04.

G[13] NAVATHE, S. B.; AND FRY, J. P., "Restructuring for large data bases: three levels of abstraction," *ACM, TODS*, to appear in June 1976.

[G14] TEOREY, T. J.; AND DAS, K. S., "Application of an analytical to evaluate storage structure", Data Translation Technical Report No. 76DE 7.1. Univ. of Michigan Graduate School of Business Administration, Ann Arbor, 1976.

(I) Introductory

[I1] LYON, J. K., *Introduction to data base design*, Wiley Interscience, Div. of John Wiley and Sons, New York, 1971.

[I2] BACHMAN, C. W., "Data structure diagrams," *SIGBDP: Data Base* **1**, 2 (1969).

[I3] BYRNES, C.; AND STEIG, D., "File management systems: a current summary," *Datamation* **15**, 11 (Nov. 1969).

[I4] OLLE, T. W., "MIS: data bases," *Datamation* **16**, 15 (Nov. 1970).

[I5] DIXON, PAUL, "The role of data management in management information systems," *IAG Journal* **3**, 2 (August 1970).

[I6] CODASYL SYSTEMS COMMITTEE, "Introduction to 'feature analysis of generalized data-base management'," *Comm. ACM* **14**, 5 (May 1971), 308–318.

(M) Data Models—Theory

[M1] CODASYL DEVELOPMENT COMMITTEE, "An information algebra, phase I report of the Language Structure Group," *Comm. ACM* **5**, 4 (April 1962), 190–204.

[M2] CHILDS, D. L., "Feasibility of a set-theoretic data structure: a general structure based on a reconstituted definition of relation," *Proc. IFIP Congress 1968*, North-Holland Publ. Co., Amsterdam, The Netherlands, 1968, pp. 420–430.

CHILDS, D. L., "Description of a set-theoretic data structure," *Proc. AFIPS Fall Jt. Computer Conf.*, 1968, AFIPS Press, Montvale, N.J., 1968, pp. 557–564.

[M3] CODD, E. F., "A relational model of data for large shared data banks," *Comm. ACM* **13**, 6 (June 1970), 377–87.

[M4] HARDGRAVE, W. T., "A technique for implementing a set processor," *Proc. ACM SIGMOD/SIGPLAN Conf. on Data Abstraction Definition and Structure*, 1976, Taylor and Ledgard, (Eds.).

(PL) Programming Languages

[PL1] CLIPPINGER, R. F., "FACT a business compiler description and comparison with COBOL and commercial translator," in *Annual Review in Automatic Programming* **12**, 1, Pergamon Press, pp. 231–292.

[PL2] INTERNATIONAL BUSINESS MACHINES, *General Information Manual—IBM Commercial Translator*, Form F28-8043, IBM Corp., 1960.

[PL3] GENERAL ELECTRIC, "GECOM—the general compiler," CPI3 144 (IOM-4-61), General Electric Corp. Computer Dept., April 1961.

[PL4] PERSTEIN, M. H., *The JOVIAL J3 Grammar and Lexicon*, Tech Memo No. TM555/002/04, System Development Corp., Santa Monica, Calif., 1965.

[PL5] SAMMET, J. E., "Base elements of COBOL 61," *Comm. ACM* **5**, 5 (May 1962), 237–253.

[PL6] CODASYL DATABASE TASK GROUP, "COBOL extensions to handle data bases," (PB 177 682), Jan. 1968.

[PL7] COBOL, *American National Standard Programming Language* COBOL, X3.23—1974, American National Standards Institute, Inc., New York, 1974.

(S) DBMS Specifications

[S1] CODASYL DATA BASE TASK GROUP, *October, 1969 Report*, (superseded by 1971 DBTG Report, currently out of print).

[S2] CODASYL DATA BASE TASK GROUP, *April 1971 Report*, (available from ACM).

[S3] CODASYL DATA DESCRIPTION LANGUAGE COMMITTEE, *CODASYL Data Description Language Journal of Development*, (June 1973), NBS Handbook 113, (Jan. 1974).

(SL) Stored-Data Definition

[SL1] SMITH, D. C. P., "An approach to data description and conversion," PhD Thesis, Moore School Report 72-20, Univ. of Pennsylvania, Philadelphia.

[SL2] CODASYL STORAGE STRUCTURE DEFINITION LANGUAGE TASK GROUP, *Design Objectives for a Storage Structure Definition Language*, 1970. (See [SL5]).

[SL3] STORAGE STRUCTURE DEFINITION LANGUAGE TASK GROUP (SSDLTG) of CODASYL Systems Committee,
FRY, J. P. "Introduction to storage structure definition."
MCGEE, W. C. "Informal definitions for the development of a storage structure definition language."
YOUNG, J. W., JR., "A procedural approach to file translation."
SIBLEY, E.; AND TAYLOR, R. "Preliminary discussion of a general data to storage structure mapping language," *Proc. 1970 ACM SIGFIDET Workshop on Data Description and Access*, pp. 368–380.

[SL4] TAYLOR, R. W., "Generalized data-base management system data structures and their mapping to physical storage," PhD Thesis, Univ. of Michigan, 1971.

[SL5] FRY, J. P.; SMITH, D. C. P.; AND TAYLOR, R. W., "An approach to stored-data definition and translation," *Proc. of 1972 ACM SIGFIDET Workshop on Data Description Access and Control*, pp. 13–55.

[SL6] BACHMAN, C. W., "The evolution of storage structures," *Comm. ACM* **15**, 7 (July 1972), 628–634.

[SL7] SIBLEY, EDGAR H.; AND TAYLOR, ROBERT W., "A data definition and mapping language," *Comm. ACM* **16**, 12 (Dec. 1973), 750–759.

(T) DBMS Texts

[T1] MARTIN, J., *Computer data-base organization*, Prentice-Hall, Englewood Cliffs, N.J., 1975.

[T2] DATE, C. J., *An introduction to data-base systems*, Addison-Wesley, Reading, Mass., 1975.

[T3] CAGAN, C., *Data management systems*, Melville Publ. Co., Los Angeles, Calif. 1973.

[T4] LEFKOVITZ, D., *Data management for on-line system*, Hayden, Publ. Co., Rochelle Park, N.J., 1974.

(U) Surveys

[U1] MINKER, JACK; AND SABLE, JEROME, "File organization and data management," in *Annual review of information science and technology*, Vol. 2, John Wiley & Sons, New York, 1967, pp. 185–222.

[U2] CODASYL Systems Committee, *A survey of generalized data base management systems*, (PB 203142), May 1969.

[U3] FRY, J. P., et al., "Data management systems survey," MITRE Corporation Report, MTP 329, (AD 684 907) Jan. 1969.

[U4] MINKER, J., "Generalized data management systems—some perspectives," Univ. of Maryland Computer Science Center Technical Report, Dec. 1969.

[U5] FRY, J. P., AND GOSDEN, J. A., "Survey of management information systems and their languages," in *Critical factors in data management*, F. Gruenberger, (Ed.), McGraw-Hill Book Co., 1969, pp. 41–55.

[U6] GUIDE INT., "Comparison of data-base management systems," Oct. 1971.

[U7] CODASYL SYSTEMS COMMITTEE, *Feature analysis of generalized data base*

Management Systems, May 1971, (available ACM).

[U8] KOEHR, G. J., et al., "Data management systems catalogue," The MITRE Corp., Technical Report MTP 139, Jan. 1973, (available from The MITRE Corp).

(V) Vendor Systems

APPLICATIONS SOFTWARE, INC.
Corporate Offices
21515 Hawthorne Boulevard
Torrance, Calif. 90503

[V1] SUNDEEN, D. H., *AS-IST—a general purpose management system*, Application Software, Inc., San Pedro, Calif., 1968.

BURROUGHS CORPORATION
Burroughs Place
Detroit, Mich. 48232

[V2] BURROUGHS CORPORATION, "B6700/B 7700 DMS II data and structure definition language (DASDL) reference manual," April 1974.

BURROUGHS CORPORATION, "B6700/B7700 DMS II host language interface reference manual," April 1974.

CINCOM SYSTEMS, INC.
2300 Montana Avenue
Cincinnati, Ohio 45211

[V3] CINCOM SYSTEMS, "TOTAL/7 reference manual—application programming," Pub. #P02-1321-2, June 1974.

CINCOM SYSTEMS, "TOTAL/7 reference manual—data base administration," Pub. #P02-1322-2, June 1974.

COMPUTER CORPORATION OF AMERICA
575 Technology Square
Cambridge, Mass. 02139

[V4] COMPUTER CORPORATION OF AMERICA, "CCA 204 data base management software system user language reference manual," March 1974.

CONTROL DATA CORPORATION
8100 34th Avenue South
Minneapolis, Minn. 55420

[V5] CONTROL DATA CORPORATION, MARS VI *reference manual*, Pub. #17305100, CDC, 1974.

CONTROL DATA CORPORATION, MARS VI *reference manual (full inversion)*, Pub. #17313000.

CONTROL DATA CORPORATION, MARS VI *reference manual (partial inversion)*, Pub. #60385900.

[V6] CONTROL DATA CORPORATION, "Query update version 2.0 reference manual," Pub. #60307500.

CONTROL DATA CORPORATION, "Data definition language for query/update subschema," Pub. #60359200.

CULLINANE CORPORATION
One Boston Place
Boston, Mass. 02108

[V7] CULLINANE CORPORATION, "Integrated database management system program and reference."

DIGITAL EQUIPMENT CORPORATION
146 Main Street
Maynard, Mass. 01754

[V8] DECSYSTEM 10, "Data base management system programmer procedures manual," DEC-10-APPMA-B-D, 2d ed.

DECSYSTEM 10, "Data base management system data base administration procedures manual," DEC-10-AAPMA-B-D, 2d ed.

HONEYWELL INFORMATION SYSTEMS
200 Smith Street
Waltham, Mass. 02154

[V9] I-D-S/II RELATED PUBLICATIONS:
I-D-S/II programmer reference manual, DE09.
I-D-S/II data base administrator guide, DE10.
Interactive I-D-S/II reference manual, DE11.
UFAS (United File Access System), DC89.
I/O supervisor, DD82.
File management supervisor, DD45.

[V10] MANAGEMENT DATA QUERY SYSTEM (MDQS):
MDQS, User Guide, DC80.
MDQS Data Base Administrator Guide, DC81.
MDQS IV, DD92.
MDQS IV Administrator Guide, DD94.

[V11] DRESSEN, P. C., "The dataBASIC language—a data processing language for non-professional programmers," *Proc. AFIPS Spring Jt. Computer Conf.*, 1970, AFIPS Press, Montvale, N.J., 1970, pp. 307–312.

INFORMATICS
MARK IV Systems Co.
21050 Vanowen Street
Canoga, Calif. 91303

[V12] POSTLEY, J. A., "The MARK IV system," *Datamation*, **14**, 1 (Jan. 1968), 28–30.

IBM (for IBM information, see local representative)

[V13] *Information Management System/360 (IMS/360) application description manual*, IBM Form No. H20-0524.

[V14] *Information Management System/360 Version 2, general information manual*, IBM Form No. GH20-0765.

[V15] *Information Management System Virtual Storage (IMS/VS) general information manual*, IBM Form No. GH20-1260.

[V16] *Generalized information system application description manual*, IBM Form No. GH20-0179.

[V17] BRYANT, J. H.; AND SEMPLE, P., "GIS and file management," *Proc. ACM 1966 National Conf.*, ACM, New York, N.Y., 1966, pp. 97–107.

MRI Systems Corporation
Box 9968
Austin, Texas 78766

[V18] System 2000 Publications:
General information manual, G-1.
Basic *reference manual* (Includes binder), A-1.
Immediate access feature, I-1.
Cobol *procedural language interface feature*, C-1.
Fortran *procedural language interface feature*, F-1.
PL/I *procedural language interface feature*, P-1.
Report writer feature, R-I.

Philips-Electrologica, B. V.
PO Box 245
Apeldoorn, The Netherlands

[V19] Publications:
Introduction to Pholas, Pub. no. 5122 991 25221.
Pholas *sub-schema DDL and SML*, Pub. no. 5122 991 25861.
Pholas *schema DDL and SSL*, Pub. no. 5122 991 25841.

Shipping Research Services, Inc.
205 S. Whiting Street
Alexandria, Va. 22304

[V20] Shipping Research Services, Inc., "The data base system Sibas: an introduction," 1974.
Aschim, F. F.; and Boone, P., "Sibas—an implementation of the CODASYL data base concept," *Management Informatics* **2**, 3 (1973).

Software ag
Reston International Center
11800 Sunrise Valley Drive
Reston, Va. 22091
61 Darmstadt
Hilpertstrasse 20
West Germany

[V21] Adabas *introduction*; Adabas *reference manual*; and Adabas *utilities manual*, Software ag of North America, Reston, Va.

Sperry Univac
PO Box 500
Blue Bell, Pa. 19422
ATV House
17 Great Cumberland Place
London W1, England

[V22] "Univac 1100 Series, Data Management System (DMS 1100) schema definition, data administrator reference," Sperry Rand Corp., 1972, 1973.
"Univac 1100 Series, Data Management Systems (DMS 1100) American National Standard Cobol (Fieldata), data manipulation language, programmer reference," Sperry Rand Corp., 1972.

Xerox Corporation
701 South Aviation Boulevard
El Segundo, Calif. 90245

[V23] "Xerox Extended DMS Sigma 6/7/9," Reference Manual, 903012B, February 1974.

Set Theoretic Information Corporation
117 N. 1st Street
Ann Arbor, Mich. 48104

[V24] Set Theoretic Information Corporation, "STDS/I reference guide," 1975.

(W) Report Generator

[W1] "Share 7090 9Pac, Part I: "Introduction and general principles," in *7090 Programming Systems, Systems Reference Library*, IBM, File 7090-28, Form JZ8-6166-1, p. 32, 1961.

[W2] Leslie, H., "The report generator," *Datamation*, (June 1967), 26–28.

[W3] Friedberg, L. M., "RPG: the coming of AGE," *Datamation*, (June 1967), 29–31.

[W4] Longo, F., "Surge: a recording of the Cobol merchandise control algorithm," *Comm. ACM* **5**, 2 (Feb. 1962), 98–100.

(X) DBMS prior to 1968

[X1] Miller, L.; Minker, J.; Reed, W.; and Shindle, W., "A multi-level file structure for information processing," *Proc. Western Jt. Computer Conf.*, 1960, Spartan Books, New York, 1960, pp. 53–59.

[X2] Green, B. V.; Wolf, A. K.; Chomsky, C.; and Laughery, J., "Base-ball, an automated question-answer," *Proc. Western Jt. Computer Conf.*, May 1961, Spartan Books, New York, 1961.

[X3] Vesper, N. R., *Information Retrieval Program*, Report C-1210 (David Taylor Model Basin), May 1961.

[X4] Postley, J. A.; and Buetell, T. D., "Generalized information retrieval and listing system," *Datamation* **4**, 12 (Dec. 1962), 22–26.

[X5] *User's Manual for TUG-Format Table Tape Updater and Generator*, Naval Command Systems Support Activity, prepared by International Business Machines Corp., Rockville, MD. Oct. 1962.

[X6] Collilla, R. A.; and Sams, B. H., "Information structure for processing and retrieving," *Comm. ACM* **5**, 1 (Jan. 1962), 11–15.

[X7] Cheatham, T. E., Jr.; and Warshall, S., "Translation of retrieval requests couched in a 'semiformal' English-like language," *Comm. ACM* **5**, 1 (Jan. 1962), 34–39.

[X8] Climenson, W. D., "Recol—a retrieval command language," *Comm. ACM* **6**, 3 (March 1963), 117–122.

[X9] Naval Command Systems Support Activity, "User's manual for the 704/7090 information retrieval," NAVCOSSACT Document No. 10S001, CM-76, Nov. 1963.

[X10] *Intelligence Data Processing System Formatted File System*, U.S. Navy Fleet Intelligence Center and Intelligence Systems Dept. IBM Federal Systems Div., May 1963.
Vol. 1. *Program description*
Vol. 2. *Program flow diagrams and listings*

Vol. 4. *Information system design and utilization*
Vol. 5. *Information retrieval.*

[X11] NAVAL COMMAND SYSTEMS SUPPORT ACTIVITY, "User's manual for NAVCOSSACT information processing system phase I library maintenance system," NAVCOSSACT Document No. 88M008, CM-52, August 1963.

[X12] NAVAL COMMAND SYSTEMS SUPPORT ACTIVITY, "User's manual for NAVCOSSACT information processing system phase I," NAVCOSSACT Document No. 90S003A, CM-51, July 1963. Supplement I published Jan. 1964.

[X13] SYSTEM DEVELOPMENT CORP., "System design specifications for LUCID phase I," Tech. Memo No. TM-1749/000/00, Santa Monica, Calif., Jan. 1964.
Vol. 1. LUCID control system design
Part 1. *The Master Tape*, Tech Memo No. TM-1749/101/00.
Part 2. *Parameter Load*, Tech Memo No. TM-1749/102/00.
Part 3. *Operational Control*, Tech Memo No. TM-1749/103/00.
Part 4. *Test Set-Up*, Tech Memo No. TM-1749/104/00.
Vol. 2. "GENDARME data processing facilities," Tech. Memo No. TM-1749/201/00.
Vol. 3. "LUCID program design: the grammar of OPAQUE," Tech. Memo No. TM-1749/301/00.

[X14] BRYANT, J. H., "AIDS experience in managing data-base operation," *Proc. of the Symposium on Development and Management of a Computer-Centered Data Base*, A. Walker, (Ed.), System Development Corp., Santa Monica, Calif., 1964, pp. 36–42.

[X15] BACHMAN, C. W.; AND WILLIAMS, S. B., "A general purpose programming system for random access memories," *Proc. AFIPS Fall Jt. Computer Conf.*, 1964, Vol. 26, Spartan Books, New York, 1964, pp. 411–422.

[X16] NAVAL COMMAND SYSTEMS SUPPORT ACTIVITY, "User's manual 1401 TUFF tape updater for formatted files," NAVCOSSACT Document No. 90S012W, CM-108, May 1964.

(X17] NMCS INFORMATION PROCESSING SYSTEM (NIPS), IBM 1410, NMCS Support Center, Washington, D.C., 1964.

[X18] INTEGRATED DATA STORE—A NEW CONCEPT IN DATA MANAGEMENT, Publication CPB-483 (5C10-16), General Electric Co.

[X19] NAVAL COMMAND SYSTEMS SUPPORT ACTIVITY, "7090 information processing system revised," NAVCOSSACT Document No. 90M002, 0M-01, Oct. 1965.

[X20] NAVAL COMMAND SYSTEMS SUPPORT ACTIVITY, "User's manual for 704/7090 TUFF MOD III tape updater for formatted files," NAVCOSSACT Document No. 10S001, CM-74, Nov. 1963. Change 1 published Feb. 1964. Change 2 published August 1965.

[X21] GRANT, E., LUCID *User's Manual*, Tech Memo No. TM-2354/001, System Development Corp., Santa Monica, Calif. June 1965.

[X22] SPITZER, J. F., et al., "The COLINGO system design philosophy," in *Information System Sciences, Proc. of the Second Congress*, 1965, Spartan Books, New York, 1965, pp. 36–39.

[X23] SDS MANAGE REFERENCE MANUAL, Publication 90-10-46A, Scientific Data Systems, May 1966.

[X24] CONNORS, T. L., "ADAM—a generalized data management system," *Proc. AFIPS Spring Jt. Computer Conf.*, 1966, Vol. 28, Spartan Books, New York, 1966, pp. 193–203.

[X25] A USER'S GUIDE TO THE ADAM SYSTEM, MTR-268, MITRE Corp., (AD 664 332), August 1966.

[X26] IDHS 1410 FORMATTED FILE SYSTEM: FILE MAINTENANCE AND FILE GENERATION MANUAL, Defense Intelligence Agency, DIAM-65-9-1, August 1966. Also, IDHS 1410 FORMATTED FILE SYSTEM: RETRIEVAL AND OUTPUT MANUAL, DIAM-69-9-2.

[X27] A DESCRIPTION OF THE INTERNAL OPERATIONS OF THE ADAM SYSTEM, MTR-216, MITRE Corp., (AD 660 581), August 1966.

[X28] DODD, G. G., "APL—a language for associative data handling in PL/1," *Proc. AFIPS Fall Jt. Computer Conf.*, 1966, Vol. 29, Spartan Books, New York, 1966, pp. 667–684.

[X29] VORHAUS, A.; AND MILLS, R., *The Time-Shared Data Management System: A New Approach to Data Management*, Tech Memo SP-2747, System Development Corp., Santa Monica, Calif. 1967.
WILLIAMS, W. D.; AND BARTRAM, R. C., *COMPOSE/PRODUCE: A User-Oriented Report Generator Capability Within the SDC Time-Shared Data Management System*, Tech Memo SP-2634, System Development Corp., Santa Monica, Calif. 1967.

[X30] STEIL, G. P., "File management on a small computer," *Proc. 1967 AFIPS Spring Jt. Computer Conf.*, Spartan Books, New York, 1967, pp. 199–203.

[X31] DIXON, PAUL J.; AND SABLE, J., "DM-1—A generalized data management system," *Proc. AFIPS Spring Jt. Computer Conf.*, (30), 1967, 185–198.

[X32] NAVAL COMMAND SYSTEMS SUPPORT ACTIVITY, "User's manual for information processing system phase 3A for the AN/FYK-1 (V) data processing set," NAVCOSSACT Document No. 88M901A, CM-123, Revision 5, August 1967.

[X33] AFLC/ESD/MITRE, *Advanced Data Management* (ADAM) *Experiments*, Final Report, (AD 648 226), Feb. 1967.

[X34] BROWN, R.; AND NORDYKE, G. P., "ICS an information control system," *Proc. IFIPS Conf. Mechanized Information Storage, Retrieval and Dissemination*, 1967, North-Holland Publ. Co., Amsterdam, The Netherlands, 1967.

[X35] *Data Language No. 1 (DL-1) Encyclopedia Pub. 2541-F*, North America Aviation,

Inc., and International Business Machines Corp., 1967.

[X37] GILDEA, R. A., *Evaluation of* ADAM *an advanced data management system*, MITRE Corp., (AD 661 273), May, 1967.

(Y) DBMS 1968 to Present

[Y1] BLEIER, R. E., *Treating Hierarchical Data Structures in the SDC Time-Shared Data Management System (TDMS)*, Tech Memo Sp-2750, System Development Corp., Santa Monica, Calif., 1968.

[Y2] BLEIER, R. E.; AND VORHAUS, A., *File Organization in the SDC Time-Shared Data Management System (TDMS)*, Tech Memo SP-2750, System Development Corp., Santa Monica, Calif., 1968.

[Y3] RAUCHER, V.; AND SCHWIMMER, H. S., *The Time-Shared Data Management System (TDMS), Language Specifications*, Tech Memo TM-3370, Systems Development Corp., Santa Monica, Calif., 1968.

[Y4] *SDS/9 SERIES Manage*, Publication No. CB 10035, Scientific Data Systems, 1968.

[Y5] ATLEE, E. S., et al., "COGENT III functional specifications," Computer Sciences Corporation, 1968.

[Y6] WELSH, W. A., "Engineered design of EDP systems," *Systems and Procedures Association Internal Meeting*, October 1968.

[Y7] "Remote file management system (RFMS), *"Computation Center Technical Staff Documentation*, Publications 0 to 14, Univ. of Texas at Austin, 1968.

[Y8] MANGOLD, C. A., "COBOL data management system (CDMS) briefing," *Proc. Guide*, **30**, (May 1970), 175–729.

[Y9] MCELROY, D. C., "The SERIES data management system," *Datamation* **16**, 4 (April 1971), 131–136.

[Y10] NAVAL COMMAND SYSTEMS SUPPORT ACTIVITY, "Information processing system (IPS) user's guide," NAVCOSSACT Document No. 88M904, TR-03, September 1971. Change 1 published Feb. 1972. Change 2 published Feb. 1973.

[Y11] MEINERS, E. E., "A machine-independent data management system" *Datamation*, **19**, 6 (June 1973), 92–98.

[Y12] NMCS INFORMATION PROCESSING SYSTEM 360 FORMATTED FILE SYSTEM (NIPS FFS), NMCS Support Center, CSM UM 15-74 October 1974).
Vol. I: Introduction to file concepts.
Vol. II: File structuring (FS).
Vol. III: File maintenance (FM).
Vol. IV: Retrieval and sort Processor (RASP).
Vol. V: Output processor (OP).
Vol. VI: Terminal processing (TP).
Vol. VII: Utility support (UT).
Vol. VIII: Job preparation.
Vol. IX: Error codes.
TR 54-74: Installation of NIPS 360 FFS.

(Z) Relational Systems

[Z1] GOLDSTEIN, R. C.; AND STRNAD, A. L., "The MACAIMS data management system," *Proc. 1970 ACM-SIGFIDET Workshop on Data Description and Access*, Nov., 1970, pp. 201–229.

[Z2] MCINTOSH, S.; AND GRIFFEL, D., "Data management for a penny a byte," *Computer Decisions*, (May 1973).

[Z3] WHITNEY, V. K. M., "RDMS: a relational data management system," *Proc. Fourth Internatl. Symposium on Computer and Information Sciences (COINS IV)*, Dec. 1972, Plenum Press, New York, 1972

AVAILABILITY OF REFERENCES

Addresses	Publications
EDP Analyzer Canning Publications, Inc. 925 Anza Avenue Vista, Calif. 92083	EDP Analyzer
ACM Association for Computing Machinery 1133 Avenue of the Americas New York, N.Y. 10036 (212) 265-6300	SIGBDP DBTG Specifications CODASYL Systems Committee Report SIGMOD Proceedings SIGFIDET Proceedings *Comm. ACM* *J.ACM* *TODS* Very Large Data Base Proceedings
Management Information Systems Research Center Graduate School of Business Administration University of Minnesota Minneapolis, Minn 55455	MISRC Publications
IFIP Administrative Data Processing Group 6 Stadhouderskade Amsterdam 1013, The Netherlands	CODASYL System Committee DBTG Specification *IAG Journal*

Addresses	Publications
Technical Services Branch Department of Supply and Services 88 Metcalfe Street Fifth Floor Ottawa, Ont., Canada KIA OS5	CODASYL COBOL Specification
British Computer Society 29 Portland Place London W1, England	CODASYL System Committee DBTG Specifications
National Technical Information Service 5285 Port Royal Road Springfield, Va. 22151	Documents with AD or PB numbers
SHARE Inc. One Illinois Center 111 E. Wacker Drive Suite 600 Chicago, Ill. 60601	SHARE Proceedings
GUIDE Int. Mr. Sandy Hill Smith, Bucklin, and Associates 111 E. Wacker Drive Chicago, Ill. 60601	GUIDE Proceedings
System Development Corp. 2500 Colorado Boulevard Santa Monica, Calif.	SDC Technical Reports, Memorandums
The MITRE Corp. Bedford Operations Box 207 Bedford, Mass. Washington Operations Westgate Research Park McClean, Va. 22101	MITRE Technical Reports