A broad range of commercial and research data base systems are analyzed. Common characteristics are discussed. These systems, which have roots in older filing systems and in punched card systems, are grouped into the three categories of hierarchic, network, and single-level models. Also presented is work on the standardization of data base systems. Research toward the discovery of new commonalities is also discussed. This paper is based on an extensive published literature.

Data structures and data accessing in data base systems past, present, future

by M. E. Senko

During the time since the publication in this Journal of a paper on the history and status of existing data base systems,[1] much has been written about data base systems in the periodical literature and in the proceedings of conferences. However, there has not been time to compile this information into a critically written book. This article cannot replace such a book, but it is hoped that it will aid the reader in developing a critical viewpoint of general trends. In addition, it provides a large number of references for one who wishes to become acquainted with recent trends in greater detail.

recent progress

Depending on point of view, there has been either little or great progress during the past four years. From a qualitative viewpoint, there has been very little progress in commercial data base technology. The four-year period has been a time of consolidation and incremental improvement. From a quantitative viewpoint, on the other hand, there has been very great progress—the number of installed generalized data base systems has increased from a few hundred to several thousand.

During this time, data base systems have become a main topic of discussion in the computer industry. The interest is, however, not due simply to the large number of installations. The fastest computers and the largest storage devices are barely able to meet the requirements posed by recent integrated systems for reservations and for management control. Much of the present interest has arisen because effective and efficient implementation of these systems represents an outstanding technical challenge in all phases of computer systems work. Finally, in the research area, a greater understanding has been developed. Significant problems, however, must still be solved to assure a worthwhile, compatible transfer of this knowledge to installed systems.

The early sections of this review place each of these points in a long-term perspective by discussing the technical evolution of the data base systems area. Later sections provide an introduction to current publications in the area, along with comments designed to guide the reader in correlating the sometimes diverse terminology. The Data Independent Accessing Model (DIAM) is used to provide a focus for this material. The review ends with a discussion of possible future developments.

information systems and efficiency

There are many proposed definitions for *information systems*, none of which satisfies all people. We simply note that a major purpose of any information system is to provide a relatively exact and efficient model of the significant resources of a real world enterprise. In this description, the criterion of efficiency is particularly important because it provides the driving force behind much of the work on computerized information systems. In the case of computerized information systems, there are at least three components in the efficiency criterion. One is the efficient use of the resources modeled by the system, such as the parts, the airline seats, the money, the people, and so forth. In recent years, one of the main places where efficiency has been gained is in the timely use of expensive resources (for example, airline seats). In many instances, timely use generates a requirement that the enterprise models follow the real-world situation second by second.

In the computer industry proper, efforts on data base systems have been directed primarily at the other two components of efficiency; computer efficiency and human efficiency. The industry has, of course, always emphasized the efficient use of computing resources. However, there has been a continued steady decrease in hardware prices and a parallel increase in salaries. Boehm,[2] for example, indicates that as much as seventy-five percent of Air Force computing costs are in the cost of software, and he estimates that this figure will rise to ninety percent by 1985. In this environment, efficient utilization of human resources is becoming a dominant issue.

Early data base systems

The first major step in the mechanization of information systems came with the advent of punched card machines. The increased efficiency and accuracy of machine-prepared reports was dramatic. These benefits, however, caused us to overlook the fact that the systems required a new type of human effort – the effort to design and implement an efficient machine accounting system. Punched card machines can work on fixed-length fields only, and are efficient only in sequential processing. To utilize these machines, the user has to spend a considerable amount of

time fitting information into fixed-length card fields and defining the processing in terms of card sorts, reproductions, and tabulations. Although this time is generally small when compared to the time gained by mechanization, it has still been a significant new cost.

With the appearance of the stored program computer, physical and mechanical efficiency again increased dramatically. In addition, the computer had a potential for flexible and powerful logical processing. This processing potential was not initially realized because processing algorithms were lifted almost intact from punched card systems.

While total system efficiency increased again, an additional housekeeping burden for fitting the problem to the computer was placed on the user. For example, the user had to write programs in bit-oriented terms for the computer to understand them. In many senses, the substantive information processing—which is the real reason for the information system—became almost completely submerged in the housekeeping details of program loops and conditional transfers. Given the improvement in the costs and capabilities of mechanical computing resources, it became desirable to shift some of the burden of building information systems from the human to the computer. All that was needed were formalized algorithms that would allow the computer to accomplish housekeeping tasks efficiently.

The initial attacks on excessive or inefficient housekeeping were carried out by workers concerned with scientific computation. Major advances included the symbolic assembler, the procedural language compiler, and the operating system. With the possible exception of the operating system, each of these developments brought the description of the substantive processing back closer to the surface of the housekeeping detail. These advances were mainly concerned with reduction of housekeeping in the central processor and in main storage.

Another major problem—the housekeeping associated with file searching and processing—remained little affected. The attack on housekeeping in file searching and processing was led by a new group, people working on information systems in commercial and military areas. Their work led to nonprocedural systems like 9PAC, the Report Program Generators (RPG), and the military Formatted Files Systems (FFS). These were implemented as the IBM 7090 FFS and the 1410 FFS, along with the Control Data Corporation INFOL, and the Informatics Incorporated early versions of the MARK systems. These systems made three major steps forward. In the first place, they separated the description of the data structure from the processing program, so that the data structure could be changed without changing the

program. They provided algorithms for translating a user's nonprocedural description of a report into machine language procedures for composing the report. Also, they made the searching of tape files an implicit (or nonprocedural) process. That is, the user could simply state the conditions for a record to be retrieved, and the system would generate the algorithms to perform the search efficiently.

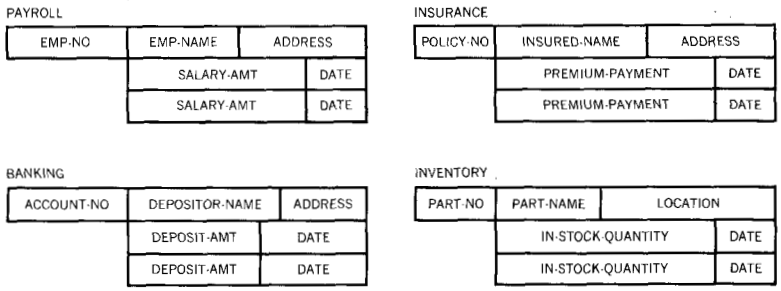
The tape-oriented systems of the 1950s and early 1960s had to generate algorithms for the search problem only within the restricted and simplified structure of a sequential search. This approach was, however, quite adequate until peripheral storage with large-scale random access capability appeared. Random access hardware removed the efficiency-based sequential batch processing restriction and made possible the construction of up-to-the-second real-world models. Real-time systems had thus arrived. Random access systems brought with them the potential for an increased range of complex file organizations. Again, to take this major step forward in total system efficiency, a significant additional housekeeping burden was placed on the user.

In the early and middle 1960s, commercial users began to accumulate a number of pragmatic techniques for shifting some of the housekeeping burden back to the computing system. The most primitive techniques were the direct, sequential, and indexed sequential access methods. These access methods assisted substantially in the housekeeping aspects of storing and moving physical records. They also provided some assistance in locating a particular record with a unique identifier.

The next qualitative step arrived with the combination of a procedural language (usually COBOL), the capabilities of early tape systems for handling records with variable numbers of segments, and the random access capabilities of hardware. This line of development, termed procedural language enhancement, appeared in the General Electric Company Integrated Data Store (IDS) and the IBM Data Language I (DL/I). Systems such as these process one record at a time and are the basis of essentially all the major real-time information systems. They generally handle real-time maintenance of the operational data of a corporation and are, therefore, called "operational systems."

Nonprocedural systems have followed a second path of evolution, by adding varying amounts of random access storage capability. This path has led to random access oriented RPGs, the MARK IV System of Informatics Incorporated, the Generalized Information System (GIS) of IBM, and the Time-shared Data Management System (TDMS) of the System Development Corporation. These executive systems, which are used primarily for data analysis, provide a major improvement in program

Figure 1 Example of a generalized data base



writing efficiency. For example, in one small, relatively informal test for reporting applications, GIS was found to require one to two orders of magnitude less programming time than COBOL. This improvement in human efficiency was provided by the executive systems with relatively small cost in computer efficiency.

The first major thrust into the real-time information systems area came about 1965. At that time each individual industry was developing its own set of management information systems. There existed approximately one hundred such IBM program products or proposed program products, each with its own specialized data management capabilities. Since the applications were certainly different, it seemed necessary for each application to have its own special code for handling its special information files. However, the IBM Federal Systems Division's Formatted File System (FFS) experience indicated that seventy to eighty percent of an application program's code consisted of file handling and data structure decoding. It demonstrated that these tasks could be handled with reasonable computer efficiency by a generalized program. It also indicated that systems could be installed perhaps fifty percent more quickly and easily if generalized code were used for communications and data handling.

The utility of generalized data base systems becomes more apparent if we look at Figure 1. In this figure, the main differences between the several files shown are merely in the names of the files and the names of the fields. Thus, such a transaction as IF_____-NO = 012345, CHANGE ADDRESS (or LOCATION) TO X could apply equally to each of the files. If the software allows the definition of records with fields of different sizes and names, and is capable of handling such a transaction with a name appropriate to one file, then it can handle similar transactions on files where the name in the blank space is different.

Perhaps the only significant differences among systems for various application areas have been the details of the computational procedures that were applied to the stored information. For

example, in the payroll case, the "salary amount" might be used in an arithmetic calculation of taxes. In the inventory case, the contents of a field similar to "in-stock-quantity" might be used in a different arithmetic calculation to obtain the number of parts to be ordered when restocking.

Realization of the fact that data handling and data communications were functions that could be generalized led IBM to emphasize the Information Management System (IMS), Customer Information Control System (CICS), and the Generalized Information System (GIS) as its main data base/data communications systems products for a wide range of industries. The historical evolution of data base systems has recently been reviewed in greater detail by Fry and Sibley.[3]

Evolution of data base terminology

One of the outstanding aspects of data base studies is the complex, overlapping, shifting, and ill-defined terminology. This situation is understandable, but its causes are not often recognized. In the data base area, as opposed to the field of mathematics, there has not been time to perfect and simplify concepts so that they can be adequately defined. In order to make some progress, complex, ill-defined concepts have been given capsule descriptions and labels such as "the sequential access method" or "secondary index" for reference. It is then hoped that the reader has enough knowledge of actual systems to understand what the label really stands for.

One of the reasons for so much overlapping terminology is that it is relatively easy to recognize a particular distinctive property of a system, assign a label to it, and then classify things according to the label. Since more terms give the terminologist a comfortable feeling that he has covered all bases, he is led to a proliferation of overlapping and inconsistent terms. Work is progressing toward simplifying data base concepts and reducing their overlap. For now, however, we shall use the labels as a starting point, leavening them with an understanding of the complex concepts in actual systems. For example, commercial data base systems are generally classified into three major categories, each labeled by a file structure, namely, the flat file, the hierarchic and the network types. These categories are rather loosely defined, and the assignment of a system to any one of them has been dependent on the time at which the system became generally known to the data base community, as can be seen by looking at the long-term evolution of this categorization.

In the beginning, there were punched cards, a box of which with the same field format provided the prototype for the *flat file*.

It was also possible to extend this structure by substituting for each card in the box two or more cards, each with a different format. This was equivalent to extending the length of a single card, because it introduced no major changes in the character of the file structure. It was also possible to extend the format by placing more than one field of a type on the card. Although this "multivalued attribute" format is really a "hierarchic form," it did not change the character of the data structure very much. Thus, these new structures were pragmatically included in the category of flat files.

A second method of extending punched card data structures was to place variable numbers of cards of a new format type after cards of the first format. Cards of the first format were called *masters* and cards of the second format were called *details*. Each set of detail cards could be considered to be physically associated with the master card that immediately preceded the set, and this physical association was used to represent a parallel logical association between the contents of the cards. This method formed the prototype for the *hierarchic file*, in which each detail card could be associated with only one master card. This master-detail characteristic distinguished hierarchic files from *network files*, in which a particular detail card or record could be associated with more than one master.

When these data structures were moved to tape systems, the physical card length constraint was removed. This new freedom placed a strain on the existing terminology. Since the term "record" could no longer stand for a physical card, what should it stand for? By analogy with the card as a physical subdivision, the physical subdivision on tape was marked by an *end-of-record gap*. The fact that it was useful to place more than one hierarchic record between record gaps brought up a problem that was resolved in some cases by calling each hierarchy a *logical record* and the space between gaps a *physical record*. There was also a problem as to what to call the logical equivalent of the single-level card. Some systems continue to call these elements records, whereas hierarchic systems frequently call them *segments*. Clearly, the data base term "record" can mean many different things.

When random access devices came into common use, the term *network* appeared on the scene. For the first time, it became useful to connect one record to a second by giving the storage address of the second record. These structures were labeled networks even though they were only a subset of the possible mathematical networks. Limitations were imposed by difficulties in storing physical records with varying numbers of pointers.

Considering the complexity underlying this simple three-part classification—flat file, hierarchic and network—we can see why

a label accompanied by a capsule description may not constitute an adequate definition. There is clearly a need for better definitions, and to some extent this need can be satisfied by appealing to mathematics. Although there are many overlapping notations in mathematics (sets, vectors, relations, etc.), it is true that when a particular notational is chosen, most users can agree on the meaning of definitions and operations within the notation. Here again one must be careful to leaven the definition with some knowledge of the technological concept. For example, an early relational definition for the term *identifier* was the following: any set of columns in the relation that provides a unique identification for each n-tuple. This definition implied that an identifier could be discovered by scanning the relation. Under this incomplete definition, weight of person would be termed an identifier when each person in the relation happened to have a unique weight.

This situation was somewhat improved by saying that over all time the set of columns must uniquely identify the n-tuples. This is better, but it still leaves out an essential aspect of the technological concept of identifier—the intention of the user to control the assignment of identifier values so that no two entities have the same identifier. (At present, the latter is an extra-mathematical part. It is essential, however, because some set of columns might by accident act like an identifier, but it would never be considered by the users as an identifier.)

This brings to the fore the distinction between the definitions of precision and accuracy. Physical scientists and statisticians know that there is an enormous difference between the two concepts. Mathematical formulations are almost always precise, but they are not always accurate representations of complex concepts. (It should be noted that it is usually better to be generally accurate than precisely wrong.) Although existing mathematical definitions are attractive to the casual reader because of their simplicity, they may not provide a useful and efficient representation of the area we wish to discuss. This is exactly the reason why new notational systems are invented in mathematics and physics, and it is also a reason for pursuing new notations in data base systems. Later in this paper, we discuss the characteristics of some of these new notations.

On the whole, there has been some progress in improving the quality of definitions in the data base area, but most concepts have not been refined to the point where simple definitions are adequate. The reader should always recognize that the concept that underlies a particular term may be more complex than the simple description that accompanies the term.

Table 1 Classes of data base systems

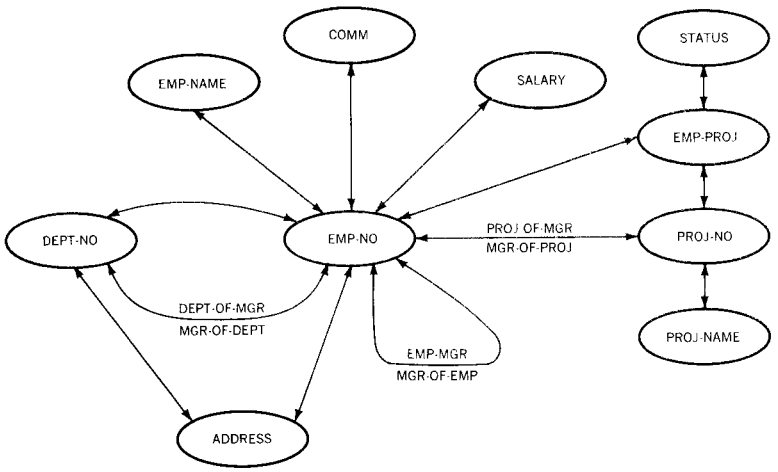
System Class	*Developer*	*Operational Systems*	*Executive Systems*
Hierarchic	ASI		ASI-ST
	IBM	IMS/VS	GIS
		DL/I	
		PARS	
	Informatics		Mark IV
	MRI	System 2000	
Network	Cincom	TOTAL	
	Cullinane	IDMS	CULPRIT
	Honeywell	IDS	
		IDS-II	
	Philips	PHOLAS	
	Software AG	ADABAS	ADAWRITER
	Univac	DMS/90	
		DMS/1100	
Single-level	Dylakor		DYL-250
			DYL-260
	IBM	CICS	IQRP

Present data base systems

To give an overall picture of the present-day environment, it may be useful to mention and give a general correlation of the characteristics of several commercially available data base systems. A deeper critical review of data base systems is available in References 4–6, and in the documentation provided by system implementers. Listed in Table 1 are some examples of commercially available single-level, hierarchic and network systems. These systems have been placed in the categories in which the reader would usually find them in the literature.

To indicate the similarities and differences among the various data structures, it is best to use some relatively neutral, hypothetical representation as a starting point. In Figure 2, we present a simple representation of the associations among important types of entities in a corporation. The ovals contain identifier values for entities and the lines indicate how the entities are related. For example, the identifier values for employees are contained in the oval labeled EMP-NO; the identifier values for addresses are contained in the oval labeled ADDRESS. The line between ADDRESS and EMP-NO indicates that there is a relationship between employees and addresses. In this case, employees live at addresses. In some cases, things are related in more than one way, as indicated by the two lines between DEPT-NO and EMP-NO. One of these lines stands for the relation-

Figure 2 Hypothetical information structure



ship between departments and their employees; the second (uniquely labeled by DEPT-MGR) stands for the relationship between departments and their department managers (who are also employees). The next section shows the representation of these relationships in commercial systems.

hierarchic systems

Many major systems use hierarchic data structures as their basic building blocks. One of the main advantages of these structures is their inherent computer efficiency with regard to storage space and peripheral storage accesses. Another advantage is a correspondence of hierarchic records to the report structures often required by commercial enterprises. A user can frequently gain human efficiency because there exists a close match between the stored hierarchic record and the form he desires for his report.

Figure 3 gives one hierarchic view of the hypothetical structure shown in Figure 2. The hierarchic view is most efficient for looking at the data from the point of view of addresses. The EMP-NO on the left is for listing employees who live at the address; the EMP-NO on the right is for employees who work in departments located at the address. In this hierarchic structure, information about a single address can frequently be obtained with only one access to the data file. Multiple accesses are often required in other types of data structures. There are, of course, other possible hierarchic structures for the same information, some of which might be more efficient for accessing information on employees and less efficient in processing address transactions. Since these other structures would have different record identifiers and/or different segment relationships, the user would have to write different transaction programs to access them, even though he wished to obtain the same basic information.

Figure 3 Hypothetical hierarchic information structure

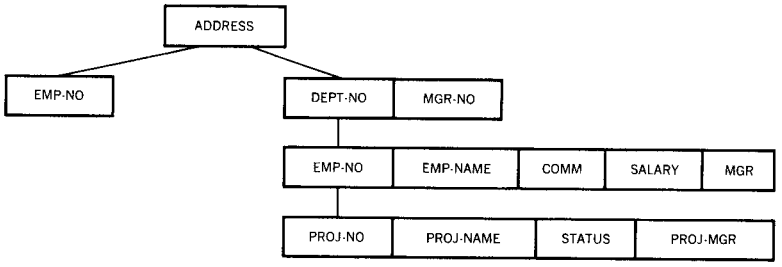
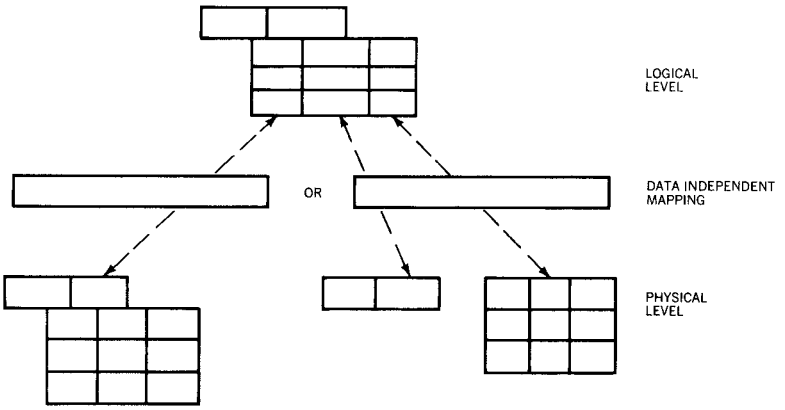


Figure 4 Mapping logical files into physical files



As we have mentioned, operational systems are usually used to maintain large data bases in real time, typically with one-record-at-a-time logic. Two noteworthy operational systems that are often classified in the hierarchic system category are the IBM IMS/VS and the MRI System 2000. In recent years, both companies have added means for pointing between hierarchic structures, so that they might now also be classified as network systems.

Looking more closely, we find that the System 2000 data structure—like that of its predecessor TDMS[7]—gives great consideration to direct-pointer-based processing of secondary index information. Both systems allow the user to define a selected number of secondary indexes. The entries in these secondary indexes point to tables that mimic the hierarchic structure of the individual data records. Internal processing of these tables allows the system to obtain a list of direct pointers to exactly those hierarchic records that meet the conditions placed on the indexed fields. The data file can then be accessed for testing on non-indexed fields and for the processing of qualified information.

With regard to data accessing, there are two types of language available: a segment-at-a-time CALL interface to COBOL, FORTRAN, PL/I, and assembler language; and a set-oriented language for accessing hierarchies. This latter language is based on the original TDMS accessing language, which in its time was one of the simplest languages available for accessing hierarchies. Recently, MRI Systems Incorporated announced a link feature that allows the user to connect multiple files by means of symbolic pointers. This feature aids the System 2000 user in performing network processing.

In using the IBM Information Management System/VS (IMS/VS),[8] the application programmer writes his programs in terms of one or more purely hierarchic logical files. The data base administrator is then separately responsible for computer efficiency. He may choose to map the logical files into actual physical files in a variety of ways without affecting the user's programs, as shown schematically in Figure 4. As a start, the IMS/VS data base administrator may choose from a spectrum of physical file organizations, depending on the ratio of sequential to direct key record accessing anticipated for the file. If the processing does not require direct accessing, insert, update, or delete calls, the administrator may choose the Hierarchic Sequential Access Method (HSAM), which relates the hierarchic segments of each record by physical contiguity and places them in a sequential file.

If some amount of direct accessing is required, but much processing is done sequentially in primary key order, the administrator may choose the Hierarchic Indexed Sequential Access Method (HISAM), which also relates record segments by contiguity, but places the records in an indexed sequential file. Excess segments of records that exceed a specified length are placed in an overflow file along with new records. In this case, the data base administrator has an alternative in the Hierarchic Indexed Direct Access Method (HIDAM), which relates subordinate segments by direct pointers. The choice between this pair depends on the amount of file reorganization that is anticipated and the speed with which the user wishes to access subordinate segments in the hierarchy. If direct key accessing predominates, but some sequential processing occurs, the data base administrator may choose the Hierarchic Direct Access Method (HDAM), supplemented by secondary indexes on the keys to be used for sequential processing. This choice can save system accesses through a primary index. If there is only direct accessing, the administrator might choose the pure Hierarchic Direct Access Method (HDAM).

If various types of segments in the logical file are accessed at greatly different rates, or if some of the lower-level segment

types are frequently processed as a separate logical file, then the data base administrator may choose to support the logical file with several separate physical files and have the system connect the files together with either symbolic (key) or direct pointers. The fact that these pointers can lead to other records gives IMS/VS network-like storage structures and processing capabilities. For example, a bill of materials for a particular part that is often implemented by network functions can appear to the IMS/VS application programmer as a logical hierarchic record.

If accesses are frequently based on a key other than the primary key, the data base administrator may also choose to have the system construct and maintain secondary index files based on these keys. Finally, the data base administrator may accommodate the system to various rates of insert-update-delete activity by varying the allocation of space in the primary and overflow portions of the supporting physical files. With the exception of secondary index changes, all these computer efficiency variations can be made without affecting the user's application program.

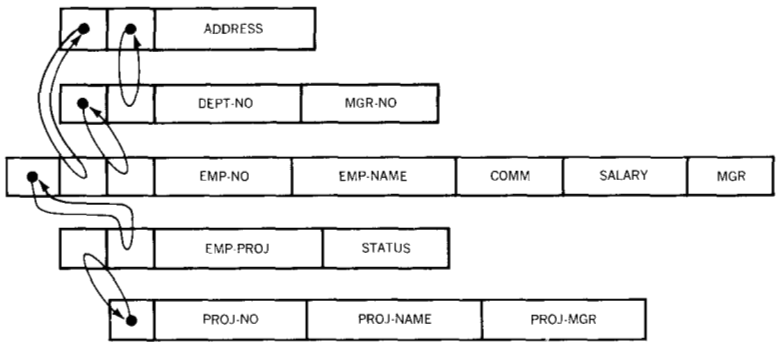
In the data accessing area, the main language of IMS/VS is the segment-at-a-time language, DL/I. The nonprocedural or set-oriented languages, Interactive Query Facility (IQF)[9] and Generalized Information System (GIS),[10] supplement DL/I to give IMS/VS executive system capability.

Finally, the IBM Programmed Airline Reservation System (PARS) and Advanced Administrative System (AAS) provide a limited-form hierarchic record structure. These extremely high-performance systems can support thousands of interactive terminals. Since, however, they are designed for special high-performance environments we treat them no further in this general review. AAS is discussed by Wimbrow,[11] and PARS is described by Siwiec.[12]

In the category of hierarchic executive systems, we place those systems or languages that process sets of records rather than a single record at a time. In addition to System 2000 with its set-oriented language, there are a number of executive language systems that can either stand alone or process files maintained by operational systems.

One well-known system in this category is MARK IV of Informatics Incorporated. MARK IV can be run as an independent system or it may be used to access files that are maintained by operational systems such as IMS/VS. When run as an independent system, the MARK IV data structures can be stored in sequential or indexed sequential files. When indexed sequential files are used, key fields may be used for direct access coordination of

Figure 5 Hypothetical network information structure



records in multiple files. The main data accessing language of MARK IV is, like the report generator languages, based on a series of forms. The user fills out these forms to specify nonprocedural file scanning, testing, and production of reports that contain totals, subtotals, etc. In recent years, Informatics Incorporated has provided a series of online terminal capabilities to assist the user in interactive access to MARK IV.

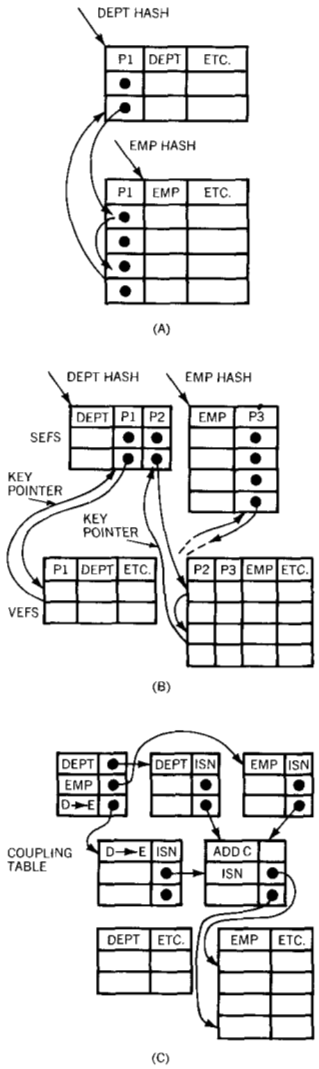
A second system in this category is ASI-ST. Like MARK IV, the ASI-ST system utilizes fixed-column forms for transaction specification in both batch- and terminal-oriented modes of interaction. It has been suggested that writing in ASI-ST could be used as a substitute for the writing of IMS/VS-DL/I procedural programs.[5] Other executive systems like MARK IV and GIS might also be used in this fashion to simplify the generation of IMS/VS application programs.

The IBM Generalized Information System (GIS) has evolved from the free-format languages of IBM Federal Systems Division's Formatted File Systems. Because of the way it evolved, GIS provides a direct terminal language. In dealing with single-level records and informal reports, GIS is relatively nonprocedural in appearance, but it does require procedural-like statements for accessing hierarchic structured records. Like MARK IV and AS-IST, GIS can be used either as an independent system or it can be used to access IMS/VS data structures. A survey of hierarchic systems has recently been published by Tsichritzis and Lochovsky.[13]

network systems

Presented in Figure 5 is one of a number of possible network representations of the example information shown in Figure 2. In Figure 5, we show only single-level records, although some network systems also allow the fixed format hierarchic structures found in COBOL record definitions. Use of this feature is, however, redundant because the network structure itself allows the specification of hierarchic structures. Since a different language

Figure 6 Three operational network systems (A) DBTG/IDS (B) TOTAL (C) ADABAS



is required for accessing the COBOL hierarchic record, this introduces another form of data structure dependence into the system.

The first major commercial network-oriented system was the Integrated Data Store (IDS) of the General Electric Company. This system was implemented in the early 1960s and was the basis for the system proposed by the Data Base Task Group (DBTG) of CODASYL. IDS has gone through many iterations of development. In Figure 6A, we present a schematic of the DBTG/IDS major file organization features. The file organization starts with a hash code accessing algorithm that assigns specified types of records to page-sized buckets on physical devices. In our example, both Departments and Employees can be accessed by hash codes. These hash-coded records can then be joined into application-oriented sets by single or bidirectional chains. In our example, the set of Employees in a particular Department are all on the same chain.

There could be two other types of pointers for this particular set. One type would also start at a particular department and go in the opposite direction, giving the second direction of a bidirectional chain. The second type would point from each individual employee record back to its department record.

A processing program usually starts by entering on a hash-coded record, and then follows chains to obtain other records as required. In our example, if the program requires information on the employee of a department, it would hash the department number to obtain the department record, and then use the chain from the department record to obtain the appropriate employee records. If the system users seldom have questions about individual employees, the data base administrator would not use a hash code to store the employee records. He would instead tell the system to have the employee records placed near (that is, in the same bucket with) the appropriate department record. Since an application programmer must know and use only the available set of physical access paths in his programs, the programs have some data structure dependence.

There is a new Honeywell version of IDS called IDS-II that contains many of the features specified by the DBTG. The major change is the addition of a subschema that allows the programmer to gain data independence by describing an application-oriented subset of the schema for his problem program. In the case of the DBTG systems, the logical subschema is limited to mapping from one schema. There are a number of other systems that are designed to implement the specifications set forth by the DBTG. These include DMS/90 and DMS/1100 of Univac, IDMS of the Cullinane Corporation, and PHOLAS of the Philips Corporation.

With regard to technical aspects, a number of possibilities for simplification of parameters have been pointed out with regard to the DBTG specifications. Such a set of simplifications was suggested at the IFIP TC-2 Conference at Wepion, Belgium.[14,15] A further simplifying process is being carried out by the CODASYL Data Description Language Commitee (DDLC).[16] This committee is classifying the complex of DBTG data description parameters into a data independent set and a set that is to be used for the computer efficiency tuning of an installation. Additional considerations with regard to the DBTG data structure concern efficiency of access to the long chains of records. One solution to the problem has been attained by including the possibility that DBTG sets can be implemented by pointer arrays. A survey of CODASYL DBTG systems has recently been published by R. W. Taylor and R. L. Frank.[17]

Two other network systems of interest are TOTAL and ADABAS. In Figure 6B, we give a simple TOTAL file organization.

Single-level records for identifier fields are stored in Single Entry Files (SEF) using a hash addressing scheme. Each identifier field value in a single entry file may have a forward address pointer to any Variable Entry File (VEF) containing that field. This pointer leads to the first record in the VEF that contains a corresponding field value. A pointer chain from this record continues through the VEF and connects all the records that contain the specific field value. There is also a chain that points in the other direction, starting with the SEF value and going to the last data record that contains that value and progressing backward through all other records that contain the same value. Records in the VEF are placed close to other records that have the same primary linkage path. The user can define this path by selecting a primary SEF from among the SEFs that have chains to the VEF. Each variable file entry also contains a symbolic (key) pointer that leads directly back to a SEF value for each chain through it. This amounts to three types of pointers for every index to the field. These pointers are analogous to the three types of pointers that may be optionally specified in the DBTG file organization, except that the backward pointer is a symbolic (key) pointer in TOTAL and a physical or logical address pointer in DBTG.

TOTAL provides hierarchic or network access paths by placing additional pointer entries in its Single Entry File. For example, if the user wants the equivalent of a master record for department and to connect this to detail records for employees, then, as in our hypothetical example in Figure 2, his Department SEF also has pointers to the Employee File. In the case of the Employee File, the pointer chains connect all the employee records that contain the same department number. To construct the hierarchic record, the TOTAL programmer then accesses the SEF

and makes as many separate calls to the employee file as there are employees in the department. In a sense, the TOTAL file organization may be compared with a DBTG organization, where record types are either masters that are accessed by hash code or details that are accessed by chains from one or more of the master files. Masters are placed in separate files from details, and no record is both a master and a detail.

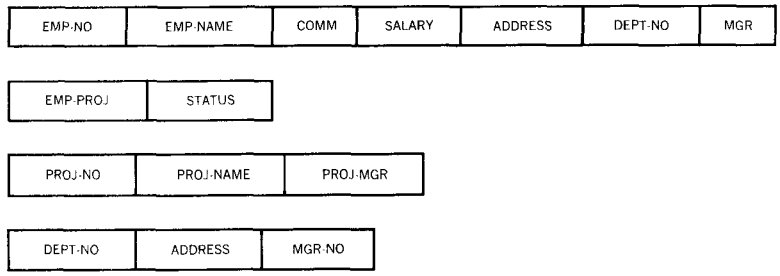
ADABAS, shown in Figure 6C, handles what are essentially flat files, but it does support multiple valued attributes and periodic fields. Using these features, a programmer can construct and decode records with more complex formats. Unlike TOTAL, it allows only one type of record per file. However, ADABAS records and fields can be of variable length because ADABAS provides both compression and encyphering algorithms. As a consequence, the programmer can gain the effect of multiple record types of defining one large record that contains all fields and then—for each particular record—entering only those fields that are appropriate to its type. There is very little space penalty for unused fields.

For each file, ADABAS maintains a list of descriptor fields, that is, fields for which secondary indexes are to be built. Each entry on this list points to a list of values for the corresponding field. Each value on this second list has a pointer to a list of Internal Sequence Numbers (ISNs) that are record numbers for records containing that value. In essence, this structure—called an *associator*—is a set of multilevel secondary indexes. Having found an ISN, ADABAS goes to an address converter that supplies the actual storage address for the ISN record.

The ADABAS coupling table provides a mechanism for network and hierarchic retrieval. A coupling table essentially acts as a connector between two files. To define a coupling table, the user selects a descriptor field that appears in both files. The coupling table for this field then provides a bidirectional mechanism for going from a record in one file to all the records in the other file that have the same value in their descriptor field.

Consider the example of hierarchic retrieval. For each department number in the department file, ADABAS goes to the coupling table to find the ISNs for records in the employee file that contain that department number. The system can then go through the address converter to obtain the addresses for the desired employee records. Here again, the system assembles hierarchic records from more than one file. There are at least three software language systems for set-oriented access to network systems—ADAWRITER, ASI-ST, and CULPRIT. ADAWRITER is designed to access ADABAS data bases in a batch mode, and there is to be a language for online interaction. ASI-ST can access both

Figure 7 Hypothetical example single-level file



TOTAL and IMS/VS files. CULPRIT provides the capability for accessing bill-of-material networks. For example, it has been written to access IDMS (a DBTG-type system) and DBOMP files. The CULPRIT language is very much like those of report program generators, in that they have fixed column entry positions.

single-level systems

In addition to the technologically prominent hierarchic and network systems, a number of systems operate solely with single-level records. That is, they provide no physical file structure assistance in relating flat files. Figure 7 illustrates a possible single-level version of the example in Figure 2. Most single-level systems would be classified as executive systems because they process multiple records at a time. Although these systems often provide simple, easy-to-use query languages, they have not usually been considered for implementation as operational systems. This may result from the difficulty they present in dealing with multiple-value relationships.

If we go back to the example of departments and employees in Figure 2, there are three ways in which this multiple-value relationship can be processed using single-level records. In the first method, a fixed number of spaces for employee numbers can be set aside in the department record (EMPFIELD1, EMPFIELD2, etc.); as many employee numbers may be entered as there are employees in a specific department. This solution has its difficulties if the user wishes to write queries with conditions on the employee fields, because there must be a copy of the condition for each field. In addition, each department record must have enough spaces available to handle the maximum number of employees that can occur in a department. This results in department records for small departments that have much waste space.

A second way is to duplicate department information for each employee. This is the solution used by the IBM MIS/360[18] and its successors. The difficulty with this solution is that the department data must be copied and maintained for each employee, a difficulty that increases if there are multiple levels in the hierarchy.

A third solution is to have separate files for department and employee records and to merge them by sorting and matching records on the value of the department number. This is the solution usually selected by report generators and relational systems. The difficulty with this solution is that it requires much processing and sorting time to make matches each time a transaction is processed. The hierarchic and network systems, of course, also make these matches, but only once at the time of the original storage of the hierarchic record. They can regain them simply by bringing in the hierarchic record when a transaction is processed.

In IBM, the most familiar terminal-oriented product for single-level records is the Interactive Query and Report Processor (IQRP),[19] which is an outgrowth of MIS/360. It has some language resemblance to the Interactive Query Facility for IMS/VS. MIS/360 is particularly interesting because of its easily used language. This language has a formal basis, but it also has several characteristics that make it appear much more English-like than most procedural languages. The MIS/360 system itself was designed to operate primarily on files that had been extracted from operational systems. These extracted files were then loaded with a selected number of secondary indexes to be used solely by MIS/360. The main difficulty with MIS/360 was that it could obtain reports from only one file at a time. Nevertheless, in terms of usability for this restricted domain, MIS/360 had an excellent language.

Because of their extensive use, Report Program Generators (RPGs) should be mentioned at this point. They are not particularly terminal oriented, but they have provided the basic form-oriented language that is a characteristic of a number of systems such as MARK IV and CULPRIT. There are, of course, a number of packages that retain the report generator philosophy and record format. Two such systems are DYL-250 and DYL-260. Finally, it should be noted that CICS, mentioned earlier, has some primitive record handling capabilities. The main purpose of CICS is, however, to handle data communications.

Standardization

There is general agreement that standardization in a stable technology can be a very good thing because standardization reduces the overall effort required to produce and use the required tools. There is also general agreement that standardization in a rapidly evolving technology can retard the growth and development of that technology because it may freeze the field in a confused and undesirable state and thereby act to discourage the development of proper tools. Each side of the issue of

whether data base systems have achieved the stability required for standardization has advocates. To understand the issues of standardization, it may be useful to mention some of the activities that have entered into the standards debate.

Data Base Task Group (DBTG)

The most prominently mentioned activity with regard to data base standards is the Data Base Task Group of CODASYL, which is an informal organization of users and producers of data systems that works to develop techniques and languages to assist in data systems analysis, design, and implementation. Reports issued by the DBTG are advisory in nature. The history of DBTG began in 1965, when an informal task group was formed to study the subject of data bases. Instead of producing a general review of the area, the committee developed specifications for an example data base system. The specifications were based primarily on two earlier systems with which the members had had experience—the General Motors Associative Programming Language and the General Electric Integrated Data Store (IDS). When the initial report was presented in 1969, IBM also submitted a proposal of specifications that included data independence, security, and integrity. The committee decided to improve the existing specifications, rather than make fundamental changes, to achieve the desired additional functions. In 1971, a revised DBTG report[20] superseded the 1969 report. The CODASYL DBTG report design has been suggested as an industry standard. In this connection, an International Standards Organization (ISO) Study Group has concluded that any standardization action in the area of data base management systems based on existing proposals is premature in the absence of criteria against which to measure such proposals.[21] On the DBTG question, IBM is not now implementing a DBTG system. IBM has recognized the need for network structures and, as we have previously noted, has provided a number of forms of support for them within data base systems such as IMS/VS.

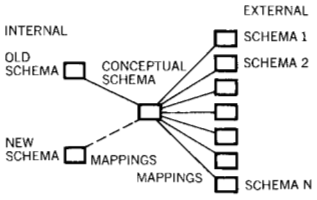
At the same time, there also exist two other organizations, the International Standards Organization (ISO), and the European Computer Manufacturers Association (ECMA). In contrast to CODASYL, ECMA does in some instances generate standards. The original charge to its Task Group Data Base (TGDB) was to report on the DB standardization options and manpower/time needed for them. TGDB, however, submitted a majority report that focused on the standardization of CODASYL DBTG.

In addition to its conclusion on standardization, ISO has accepted the Interim Report of the ANSI/X3/SPARC Study Group on Data Base Management Systems as an initial basis for discussion on a gross architecture of data base management systems.

ANSI SPARC Study Group

In late 1972, the Standards Planning and Requirements Committee of the American National Standards Institute/X3 (ANSI/X3/SPARC) established a study group to review the current state of development in the data base systems field with the objective of determining whether standardization activities were appropriate. The study group produced an overall architecture with twenty-eight interfaces.[22] We cannot discuss all these interfaces, but it might be useful to note the gross architecture for the data representations between the end user and the internal storage media. To make feasible the level of data independence and compatibility with various end user languages that it desired, the committee recognized that it would have to specify at least three major levels of data representation within its system architecture. These three levels are presented in Figure 8.

Figure 8 ANSI SPARC proposal for data base data structures and mapping



The most striking feature in the ANSI SPARC architecture is a conceptual schema level. This level is specified to provide a "data structure independent description" of the real-world enterprise. It is a formalization of an idea that could be seen in the SHARE/GUIDE Report.[23] One aspect of this idea is that the conceptual schema level should be as stable as possible to changes in the underlying physical file organizations. This gives the system data independence in the sense that programs that are written in terms of this level (or in terms of user views mapped from it) should not have to be changed when the underlying stored structures are changed for reasons of computer efficiency.

To accomplish this purpose, the conceptual-level architecture proposes that the entity classes (employees, parts, departments, etc.) that are recognized in an enterprise along with their attributes and relationships be used as points of departure for the entire data base system. A level based on such concepts should be at least as stable as, and probably much more stable than, the stored file organizations used to represent it in the computer. Stored file organizations clearly have to change whenever the entity classes and their relationships change (for example, when employees became related to departments through projects). In addition, they have to change to maintain efficiency in the face of an evolving system load, even in cases where the entity relationships do not change.

As a second requirement, the system should be able to coordinate and control all accesses to a particular stored fact (that, for example, a particular employee had a particular salary). This is a problem because there exists a recognized need to provide a series of views of the system data to the end user, both in terms of various programming languages and in terms of various specialized views of the enterprise (the personnel view, the payroll view, etc.). Mapping directly to a changing storage structure level from each of the many user views in an evolving system would become very burdensome. Essentially, a new map

would have to be made for each affected user view every time the structure changed. In addition, for each access request, the system would have to look at all maps to determine whether interference would occur.

The ANSI SPARC solution to both these problems is to have a canonical conceptual level; that is, a level on which each fact appears only once. In this solution, all the external views of a particular fact are mapped from one stable place in the conceptual level, and the system needs only to look at that place to determine whether another user is accessing the same fact. In sum, there are two definite requirements for the conceptual level: data structure stability and data sharing coordination. Steel of the ANSI SPARC Committee has presented one proposal[24] for the conceptual level that is firmly grounded in modern symbolic logic. We shall discuss other proposals in a section on logical-level models.

Existing commercial systems do not have anything that corresponds to a conceptual level. In fact, most systems have only a single level where the user deals directly with what corresponds to the ANSI SPARC internal schema. One early implementation of a two-level system was the IMS implementation of logical and physical hierarchies. The two levels in IMS correspond closely to the external and internal levels of the ANSI SPARC proposal. The DBTG schema and subschema levels also correspond closely to the external and internal schema levels of ANSI SPARC, but they do not allow the user to combine separate internal-level files.

In summary, there is considerable interest from a number of implementers for formal or informal standardization based on the DBTG report. At the same time, there appear to be major new capabilities in the offing, as exemplified by the basis for progress on a broad front as laid down by the ANSI SPARC study group.

Recent research

There are presently two divergent paths of research on data base systems—system functions and system performance. System functions research deals with data models, data access languages, and data dictionaries. System performance research deals with workload description, and with the design, simulation, and optimization of file organizations.

system functions

Present-day data base systems exhibit a strong tradeoff between simplicity and power. Either the system is simple and less powerful or it is powerful and less simple. It should be possible, however, to achieve simultaneous improvements in simplicity and

power. The way to achieve this optimization is to break up the data base problem into the right kinds of subproblems or components. To divide and conquer is the technique often used to solve complex everyday problems. In its best form, the idea is to break a problem up into two or more levels of detail that are often called "levels of abstraction." If an appropriate set of levels and components can be chosen, the user can solve certain aspects of his problem at the first level without worrying about all the details and then solve other aspects, one level at a time, by adding the details embodied at each level.

Sometimes termed structured programming or abstract data type definition, the technique has received much attention in the computer science area. In structured programming and in abstract data types, however, a new set of components is defined for every problem. The new aspect of the work in the data base systems area is that one set of components is designed to cover all applications. If this work succeeds, then the study and use of data base systems may acquire some of the discipline of chemistry and physics. A student could learn a relatively simple set of components and interaction rules—such as the elements and valence rules in the periodic table—and use them to build applications throughout his career. He would not have to learn a new complex, overlapping, inconsistent terminology for every new system, and he would not have to invent a new set of components to solve every new systems problem.

The logical and physical levels found in papers like those of Madnick[25] and Meltzer,[26] as well as in the GUIDE/SHARE Report,[23] are pragmatic examples of the abstract-level approach. The essential idea, in data base terms, is to allow the user to solve the logical aspects of his problem first, and then to take up separately the physical storage structure details needed for efficient support of his logical application structure.

Since the early papers, research has moved toward more fundamental and precise definitions of logical and physical levels and their components, and toward a better separation of functions into these levels. For example, the following four levels are defined by the present author in the original DIAM paper.[1]

- Data structure independent Entity Set Level.
- Access Path Level.
- Encoding Level.
- Physical Device Level.

In DIAM, a major effort has been made to obtain a clean separation between the entity set level and the lower physical levels. An explicit list of parameters is also given for each member of the small set of component types defined for each level.

A similar effort is being carried out by the Data Definition Language Committee of CODASYL[16] that has proposed categories that include the following:

- Schema, for those components that—for example—give the name of a schema.
- Structure, including components for describing data elements and their real-world relationships. (This category and the succeeding category correspond roughly to the DIAM entity set level.)
- Validation.
- Access control.
- Tuning (which corresponds to the DIAM access path and encoding levels).
- Resource allocation (which corresponds to the DIAM physical device level).

Until recently, most other research under the name of "data model studies" has focused on the logical level only. (In different contexts, this is called the end-user level or the Conceptual Schema Level.) In most cases, these models have retained elements from physical representations (e.g., single-level record structures) and, therefore, have not made a clean separation between logical and physical levels. Nonetheless, their intention has been to define a logical-level model, and that is the basis on which data base systems research is discussed here.

evaluating logical-level models

After much consideration and deliberation, it has been agreed that it is possible to map field values from any one logical-level model to field values in any other logical-level model (that is, between flat files, hierarchies, networks, etc.). The main differences between models occur in the ways in which they relate field values and in the number of physical structure elements they contain. Differences in relationships are important because some kinds of field relationships correlate well with relationships between entities in the user's practical model, and other types of field relationships do not. For example, in the single-level file model, each field in a record is equally related to each other field, simply by the fact that they are in the same record. In the real world, some of the relationships represented in the record may be more direct than others, and the equal representation is misleading.

In an employee file record, there is no explicit distinction made between the relationship of "salary" and "secretary" and the relationship of "employee" and "salary." Even though the users of the system know that the relationship is "salary of employee," this may not be the real-world relationship that the placement of salary in the record represents. The placement of salary in such a record might equally represent "salary of secretary." If no fur-

ther guidance is given, a user might come to the wrong conclusion about the meaning of the field.

The use of the physical concepts of files and records also places an added translation burden on the user because he must tell the system how to find and search files and records, instead of simply asking about the entities he is interested in. Finally, some types of relationships lead to easy evolution of the system model when a user's picture of the real world changes and others do not.

At first glance, issues such as these seem relatively unimportant. For example, it seems natural and necessary to translate problems into terms of files and records; therefore, we need not try to get rid of this burden. Many assembly language programmers had an analogous feeling about registers when compilers were being proposed, yet compilers have been an extremely helpful development. Similarly, making systems evolve gracefully over time is important. It is frequently suggested that fifty to seventy percent of programmer time is devoted to changing old programs to meet new circumstances. In this situation, the important issue then becomes that of how well a logical model provides the desired relationship and system properties. For example: how faithfully does a logical-level model represent real-world relationships? Does it imply spurious relationships that do not exist in the real world? How stable is the match to changes in the real world? Do the information structure and programs have to be changed substantially when a small change occurs in the real world? What is the ease of use of the accessing language?

To answer such questions, most recent work has focused on the ability of the model to represent real-world relationships and on the ease of use of possible accessing languages. Since there is no mathematical formula for evaluating models, the work tests the capabilities of models and compares different models by judging how well they work with regard to examples of possible queries and possible kinds of system evolution. Logical-level model research has focused on models with simpler basic components than the hierarchies and networks to be found in most commercial systems. In particular, recent research on logical-level models can be separated into the following two categories: (1) single-level files (termed n-ary relational systems by many authors), and (2) binary associations.

single-level file logical models

While there has been great recent interest in relational data base systems, single-level logical models have roots in the use of the punched card (which is a single-level record). One of the best-known data processing systems, the report program generator, uses a single-level logical model. For example, the operation "match" in report program generators is the same as the operation "join" in relational terminology. In effect, the basic logical models for report generators and relations are almost exactly the

same, except for terminology and accessing language.

Some of the earliest data base research work on single-level relational files was reported by McIntosh and Griffel[27] in 1968. A paper on the Entity Set Model by Davies[28] gave added impetus to research on single-level files. This paper was followed by a paper by Codd,[29] who discussed single-level files in terms of the mathematical theory of relations. Codd added a number of terms that made the theory more compatible with the properties of data processing files. His paper led to a considerable amount of work in universities and within IBM on relational data systems and languages. In general, the relational work has focused on the logical level and has not addressed the need for powerful physical file organizations at lower abstract levels to obtain reasonable system efficiency.

The initial basis for research in the single-level area was expanded by the Data Independent Accessing Model (DIAM).[1] This model more closely followed the terminology presented by Davies, Meltzer, and the later SHARE/GUIDE reports. It provided a basis for a general set of file organization techniques, including hierarchic structures and indexes for efficiently supporting the single-level entity set model. In essence, DIAM was a data model that included a logical-level model as one of its levels. Additional work provided the set-oriented language RIL[30] for accessing the entity set model and algorithms for selection of optimum paths[31] to satisfy set-oriented transaction statements.

The Entity-Relationship Model—another multilevel n-ary model— has been presented by Chen.[32] Early publications on this model were primarily concerned with the description of an improved logical level, particularly, a more detailed and flexible method of describing a network of relations. These publications have not contained any detail on the components of the lower, stored data structure levels.

implementation

Almost all research on implementation has been directed toward implementing relational language systems. Noteworthy work outside IBM has been done on the INGRES system[33] at the University of California, the ZETA system[34] at the University of Toronto, the RDMS system[35] and the RISS system[36] at the Massachusetts Institute of Technology. The underlying file organizations for these relational systems resemble the file organization for the early MIS/360. That is, they allow for any number of secondary indexes to a single level file. Although such organizations are useful for transactions that refer to a single relation, they are often inefficient for processing matches between relations. At present, interfile relationships must either be built every time a transaction that requires them is executed or when single-level files must be applied, such as when department information must be duplicated for every employee.

There is considerable work going on to improve efficiency. To approach the efficiencies of hierarchic or network systems, relational systems will have to have the ability to describe and access stored hierarchic or network structures. A paper by L. Schneider[37] has shown how this difficult capability can be achieved in a general manner by using the DIAM model. There is, in addition, work going on at the University of Toronto[38] and the University of Illinois[39] to provide limited versions of this capability, starting directly from a relational context. The DIAM model has not been implemented in the form of a data base system, but a group from the Martin Marietta Corporation has implemented a generalized system performance simulator based on its specifications. This simulator has been able to describe the System 2000 organization by using a set of DIAM parameter tables.[40]

accessing languages

The second area where relational theory helps in understanding data base systems is with regard to the structure of user languages. Either relational algebra or relational calculus can provide a formal mathematical basis for the construction of possible user-oriented languages. The following is an example query presented originally by Date.[41]

English:

"Get supplier names for suppliers who supply at least one red part."

For the relational tables:

SUPPLIER (SUP-NO, SUP-NAME, STATUS, CITY)

PART (PART-NO, PART-NAME, COLOR, WEIGHT)

SUP-PART (SUP-NO, PART-NO, QTY)

Relational calculus language:

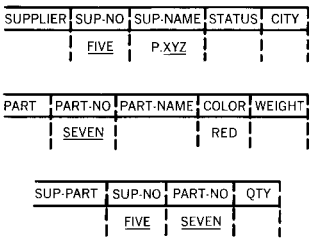
```
RANGE PART PX
RANGE SUP-PART SPX

GET W (SUPPLIER.SUP-NAME):

  ∃SPX (SPX.SUP-NO=SUPPLIER.SUP-NO ∧∃PX (PX.PART-NO
  =SPX.PART-NO ∧PX.COLOR='RED'))
```

Significant aspects of this statement are the phrases "SPX.SUP-NO=SUPPLIER.SUP-NO" and "PX.PART-NO=SPX.PART-NO". These are required to interconnect the three relations. Phrases like these appear in all record-oriented systems. It is shown later in this paper that such phrases are not required in semantic networks, thereby simplifying the writing of program statements.

It has been suggested that statements like these might be expressed in a more user-oriented relational language. Exactly what form such a language might take and whether it might differ significantly from existing nonprocedural languages—like those for MIS/360, SYSTEM 2000 and GIS—is not clear at this time. Until the projected relational language appears, it is difficult to judge the practicality of relational principles.

One strikingly different approach to single-level file languages is Query-by-Example.[42] This language, like the report generators, employs a fixed column input form. However, it has many unique features. For example, the order of statements in a transaction specification is immaterial. This relieves the user of the burden of constructing his query in a sequential fashion. Instead, the user constructs the query a line at a time in any order. The Query-by-Example statements for the previous English language and relational query are presented in Figure 9.

Figure 9 Query-by-Example



In Query by Example, we can use any symbol as an example of the element we want to talk about. Underlining indicates that the element is an example. If an element is not underlined, the symbol stands for itself. In our specification, "FIVE" is an example of a number for a supplier, and "SEVEN" is an example of a part number. Since "RED" is the actual color that the part must have, it is not underlined. The "P." before "XYZ" indicates that the supplier's name should be printed.

The Query-by-Example language has been the subject of a human factors experiment to determine its ease of use with respect to the Interactive Query Facility (IQF),[43] Tests were run with subjects who were both experienced and inexperienced in programming. For those functions that Query-by-Example could provide, it seemed easier to use. Such tests may guide us as to desirable language features, from a human factors point of view.

semantics

Much of the recent research on logical-level models has been concerned with making the models represent the semantics of the real-world situation more closely and exactly. For example, we would like to present the user with a model that will restrict him from performing nonsensical operations. This approach is to be contrasted with today's systems that present the user with a logical-level model in terms of computer stored files of records that contain bits or bytes. In these systems, the user can add any field to any other field and store the result. (For example, one can create a nonfact by adding "age 24" and "weight 150" together and storing "174" as an "address." There are already some compilers that forbid certain simple nonsensical operations like adding a floating point number to a fixed point number without conversion. Clearly, it would be useful if the system were to forbid other meaningless operations.

Figure 10 A meaningless relation

EMPLOYEE RELATION

EMPLOYEE NUMBER	NAME	ADDRESS	FRIEND	SALARY

As we mentioned earlier in this paper, the best way to evaluate models is to use examples. Kent has presented examples of meaningless operations in two recent papers.[44] Another set was presented in Bracchi, et al.[45] Also, an early paper by Codd[29] noted that certain relational operations produced results that were not meaningful in the real world. The DIAM paper[1] went further by pointing out a need for restricting information system operations to those that produced meaningful results, and also gave some examples.

More recently, Schmid and Swenson[46] have discussed similar examples in a relational framework and have pointed out a number of places where additional constraints should be placed on the relational model. Figure 10 gives one of their examples. One question posed by this example is what the appearance of FRIEND and SALARY in the same relation implies. Does SALARY imply "salary of the FRIEND" or "salary of the EMPLOYEE"? Such a relation is without semantic meaning, and something must be added to make the meaning clear to the user. In the Schmid-Swenson approach, there is an attempt to define meanings in terms of constraints as add-on features of the relational model. When examined in detail, their proposals for describing these constraints lead them to a model that is remarkably similar to the binary models to be described in a later section.

natural language models

Since other disciplines are also interested in the topic of semantics, it is useful to consider their experience. Two of the major disciplines are natural language processing and theorem proving. In each of these disciplines, it is important to treat the names for things in a manner that is meaningful in real-world terms. In addition, these disciplines study better ways of characterizing the data that they process and the operations they allow to process it.

A natural language approach using a semantic network is being followed by Roussopoulos and Mylopoulus at the University of Toronto,[29] in which they try to make constraints inherent parts of their model. Their model is a semantic network, and their work has been published both in the data base and the artificial intelligence literature. In their proposed use of the model, the user and the operations he applies to the stored data would be constrained by meanings implicit in the semantic network.

At this point, it is appropriate also to mention data base related work in natural language query systems. Much of the early work in this area has been devoted to creating systems for purposes of demonstration. In these demonstrations, many natural language queries could be parsed and answered, but the fraction of a set of queries posed by an inexperienced user that would be interpreted correctly was not clear. Systematic work on application-oriented data bases is continuing. For example, Woods[47] has produced a system to answer natural language questions about lunar rocks. Also, a group in the IBM Research Division is working on a natural language query system for urban planning.[48] Petrick has recently published an excellent discussion of the use of natural languages for communication with computers.[49]

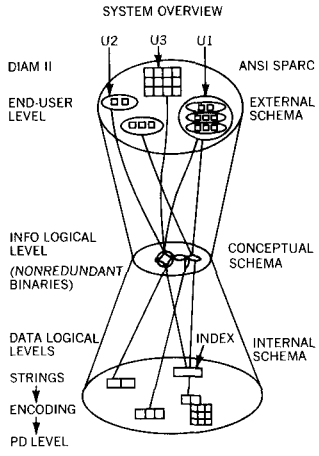
A hint of the difficulty involved in semantics can be given by the following query: "Print departments and their employees where employees earn over $20000." The question is whether the system should print "all the employees in the department," or "only those that earn over $20000," or "only departments where the sum of all employee salaries is over $20000." This query cannot be answered without additional information.

It has been suggested by a number of authors that such a system should have a dialogue with the user to obtain needed information. If the system cannot understand English, then the user must learn some formal language that the computer can deal with. This defeats the reason for using natural language in the first place. Fortunately, it appears that most queries have clearcut answers. This means that there is hope that a system could answer a sufficiently large fraction of possible queries to be useful and not give seriously misleading answers in other cases. The truth of this conjecture can only be determined in a real operating environment.

binary logical-level models

Noting that Figure 2 is also a representation of information, one might ask whether there are any systems that use such a binary-association-oriented representation directly. The fact is that that representation has many similarities to the semantic networks that are used in natural language systems. There is also a long history of work that uses binary associations in artificial intelligence research. Some of the earliest work was done on the Relational Data File by Levien and Maron.[50] Later work was done by Ash and Sibley,[51] Feldman and Rovner,[52] and others who were concerned primarily with question answering or theorem proving systems. None of these early systems has been considered for commercial use, perhaps because they store each set of binary relations in a separate file. Such a file organization is particularly inefficient when the number of individual relations is large. In this case, many accesses must be made to peripheral storage devices to process the different files.

Figure 11 Five DIAM II abstract levels for mapping binary associations at the ANSI SPARC conceptual schema level to hierarchic records at the external schema level



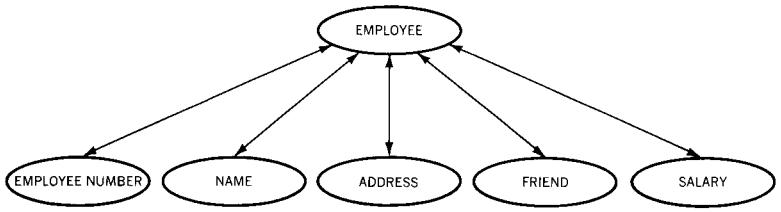
With the recent study of abstract levels, it has now become clear that an information representation can be supported by a stored data representation of a completely different form. For example, Figure 11 presents the five DIAM II abstract levels[53] for mapping binary associations at the ANSI SPARC conceptual schema level to hierarchic records at the external schema level, and to indexes, lists, and hierarchic records at the internal schema level. This mapping flexibility means that binary information representations need no longer be saddled with the inefficiencies of stored binary file organizations.

Given the promise of efficiency through mapping, interest in binary relations has been renewed. Much of this interest is because binary relations seem to be a fitting semantic representation of facts. That is, binary relations can represent only the facts that the user wants in his real-world model, and do not carry along spurious associations like FRIEND and SALARY in n-ary relations that must be removed from consideration by some add-on mechanism. Figure 12 shows a binary representation of the information listed in Figure 10. In this case, it is clear that SALARY is a direct attribute of EMPLOYEE, and it is only indirectly related to FRIEND by way of EMPLOYEE.

Langefors[54] and Sundgren[55] have created renewed interest in the area with their series of papers on structures for representing the real world. Additional impetus came from Titman's[56] and Bracchi's[57] work on binary relations, and a major force was Abrial's paper on data semantics.[58] This latter work was extended in a paper by Senko.[59] There have been few implementations of binary systems since the work of Feldman and Rovner. Titman's paper presented one implementation, and Bubenko[60] and Berild and Nachmens[61] describe a second running system.

Papers at the IFIP TC-2 Working Conference at Freudenstadt, Germany, in 1976, seemed to agree that the network, hierarchic, and single-level files brought the representation of too many individual facts into their records and caused maintenance and semantic difficulties. There was a movement expressed in papers by Bracchi, Paolini and Pelagatti,[45] Falkenberg,[62] Hall, Owlett and Todd,[63] and Senko,[53] toward a smaller binary form of fact representation. At the Freudenstadt meeting and at succeeding data base meetings, there has been increasing agreement that the binary network form or some close approximation has much more desirable technical properties than n-ary relations, tables, hierarchies, or DBTG networks for use as a logical level. However, since past experience often plays a part, tables may be more desirable from a human factors standpoint. The resolution of this dilemma should cause much lively discussion in the next few years.

Figure 12 A binary representation



An issue that has arisen recently deals with the handling of time in a data base system. Due to the technological constraints of hardware, we currently maintain data bases as time slices that contain the set of most recent changes to fields in records. Each time we modify a field, we overwrite the previous record and perhaps place a copy of it on a recovery log tape, where it is no longer accessible to normal processing. This process creates a number of problems for data sharing and recovery management.

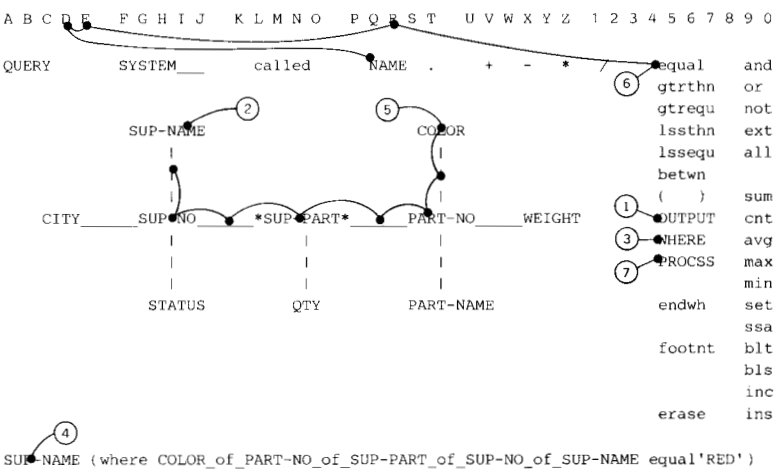
A more ideal way to view a data base system would be to consider it as a permanent repository of facts that may be valid for some time period. Thus, if a person withdrew money from his bank account on a particular date, then the fact of account balance after the withdrawal would be added to the data base with a time stamp indicating when the new balance became valid. The old balance would not be deleted; rather, it would be given a time stamp to indicate when it had become invalid. In this situation, the bank would no longer have to wait for the weekend or the end of the working day to calculate a total balance. It would simply look through its set of facts to see which customers' balances were valid at the time it wanted to check the balance. Similarly, obtaining historic information of any kind would be a normal process rather than a process of writing special programs to access old data base dumps.

A number of people are beginning to look at the technical consequences of a system that operates in this mode, as indicated by recent papers by Falkenberg,[64] Bubenko,[65] and Schueler,[66] as well as the earlier work of Davies and Bjork.[67] The general area of logical-level models has been reviewed by Kerschberg, Klug, and Tsichritzis.[68] A more recent review by Senko[69] focuses on binary association or single fact models.

accessing languages for the binary logical level

In spite of some agreement on single facts as the basic building block for the logical level, there remains the problem that many proposed languages for the binary logical level are very mathematically oriented or they deal with one binary relation at a time. In this sense, such languages are very much like the relational languages for single-level files. Their operations are specified in

Figure 13 Example FORAL light pen file query



terms of named relations, and effort must be expended to specify connections between relations (the multifile query).

This problem is alleviated to some extent in the FORAL language for the DIAM II system. FORAL operations are related to a FORAL context, and this relationship allows the system to specify what would normally be considered connections to other files in an implicit fashion. In Figure 13 is shown a display screen for a version of FORAL, called FORAL LP (Light Pen).[70] In this language, the user enters statements by touching nodes and arcs in the network and operations on the operations list. On the screen, we represent the series of pen touches required to create the example query to obtain suppliers of at least one red part presented earlier. The bottom of the screen presents a linear listing of the textual language FORAL II[71] for user feedback.

The light pen language seems to require fewer artificial elements (parentheses, commas, etc.) than a written language to render it unambiguous. This particular light pen syntax also requires little or no typing skill. Of the ten major words in the statement, nine are entered with the correct spelling by single pen touches. The word RED can either be picked out on the alphanumeric list or typed in after the light pen phase has been completed.

Hierarchic report forms that contain such things as columns that represent general expressions of attributes can also be defined naturally using either version of FORAL. For example, the FORAL statement on the left in Figure 14 gives the report on the right. Note that there is no difference in this language syntax between asking for the single-valued attribute STATUS and a

Figure 14 Example FORAL report creation

```
output                                  SUP-NO   STATUS   QTY_of          avg QTY-of
SUP-NO                                                   SUP-PART_of     SUP-PART_of
   print                                                 SUP-NO          SUP-NO
   STATUS                                  1        E        7
   QTY_of_SUP-PART_of_SUP-NO                                 9
   avg QTY_of_SUP-PART_of_SUP-NO                            14              10

                                           2        G       23              23
```

multiple-valued attribute QTY_of_SUP-PART_of_SUP-NO. In a flat file system, the user could easily get STATUS out of the SUP-NO file by asking for the appropriate field. But to get QTY_of_SUP-PART_of_SUP-NO, one would have to call for the SUP-PART file and specify a match between the SUP-NOs in the two files.

By using binary association networks, it is possible to design languages that avoid the syntactic noise found in languages that deal with flat files, hierarchic files, or DBTG networks. In doing this, we can derive languages that are both formal and unambiguous and have many similarities with natural language.

stored data levels

Most of the work on data models has been restricted to the logical level. There was, however, some excellent early work on formal models for stored data by Smith[72] and Taylor.[73] This work has been recently followed up by the CODASYL Stored Data Definition and Translation Task Group (SDDTTG), of which both Smith and Taylor are members, and which is publishing an extensive work on data translation[74] that has used and improved on many of the DIAM concepts and components. A paper by Nahourii, Brooks, and Cardenas[75] takes another approach to data translation. They are concerned about dynamic access to information stored in different generalized data base management systems. Their approach consists of describing the stored data structure in each of the systems by a DIAM string catalog. If the user specifies his query in a self-contained set-oriented language for his own system (like the set-oriented language for System 2000 or IQF for IMS/VS), then a search path selection algorithm can look at the string catalog and translate these statements into segment-at-a-time language searches for any of the system nodes that contain relevant information. Using each data base system's own segment-at-a-time language avoids the writing of translation algorithms that consider the encoding and access methods of each individual system. In another improvement, DIAM II, the model has been revised to support a binary network logical level[53] and the physical device level has been better defined.[76]

We have already mentioned several models for stored data that are being worked on in the n-ary relational environment. These

include the models of Chen,[32] Mylopoulos, Schuster, and Tsichritzis,[34] and Schmid and Bernstein.[38] Another model in this area is that of Cabanes.[77] These models tend to discuss access path structures of greater generality than those found in existing data base system implementations, but do not yet give detailed, generalized parameters for access paths, encoding and access methods. Finally, there is some recent work which uses Abrial's binary model as a basis for describing both the logical level and the access path level. A paper by Adiba and Delobel,[78] like that of Nahourii, et al., attacks the problem of cooperation between different data base systems. Both of these papers also discuss access path selection algorithms like those described by Ghosh.[31] The paper by Hainaut[79] focuses on the optimum search path selection problem.

summary: data models and accessing languages

The main trend in logical level modeling is toward a more faithful representation of the semantics of users' models of the real world. This trend has brought with it emphasis on models with simpler components than the networks and hierarchies found in most commercial systems. If followed to its apparent conclusion, the work will result in the definition of a basic data structure component for representing a single fact in the real world, rather than a complex structure containing many facts.

summary

The main trend in stored data strucures is in the other direction—away from simple tabular structures toward structures of more generality and more efficiency. This topic can be worked on in an incremental fashion, as we have seen, with the extensions of relational data structures. Also, research can start with a very general structure, as was done by Smith, Taylor, or in the DIAM model.

In the area of accessing languages, most of the work is directed toward languages that access sets of elements rather than the usual record-at-a-time languages. Here again there may be a trend toward binary-oriented languages, although Query-by-Example has demonstrated excellent usability and is a major query language innovation.

The main obstacle to set-oriented languages and the simpler logical-level models is the number of difficult research problems yet to be solved, particularly in the area of shared update and system efficiency. Although these problems will take time to solve, the various systems mentioned give an indication of the direction solutions will take.

data dictionaries

In the best of possible worlds, the functions of current data dictionaries would be integrated parts of a data base system catalog. Clearly, specifications of data elements and real-world relationships, along with their validity checks, should appear as

integrated parts of a logical-level catalog. Similarly, the supporting physical-level file organization descriptions should appear as part of an associated physical-level catalog.

It is often the case that a particular crucial need appears first in actual systems in a business installation. This need is typically met first by a special package, with research then following afterward. This seems to be the case with data dictionaries. There seems to be little direct research in the area of data dictionaries. Most of the current work is driven by user requirements, in the same way that user requirements generated development of data base management systems like IDS, IMS/VS and CICS. For example, early work in IBM on data dictionaries by Meyers on a system called TAG[80] and the more recent DB/DC Data Dictionary[81] have both grown from field experience. There are general discussions of data dictionaries by Uhrowczik[82] and Canning.[83] With regard to research, the primary need is a good data model. An appropriate integrated data dictionary should be a natural consequence of such a model.

System performance

Up to this point, we have been concerned with research on functions, research designed to improve human efficiency. On the side of the coin lie questions of machine efficiency. Questions of machine efficiency will remain as long as hardware storage is accessed by address rather than by content. The questions arise because a well-designed file organization can often provide an order-of-magnitude or greater access time reduction to desired information than a straightforward one. Since we cannot expect to see content addressing hardware that is capable of storing large data bases for an extremely long or an infinitely long time, these order-of-magnitude economies through design should continue to justify effort expended on research.

In the performance field, much of the early technology was generated by workers with backgrounds in scientific computation. Digital system simulators exemplified by the IBM Computer System Simulator (CSS)[84] and analytical simulators[85] fall into this class. These simulators describe an access to information simply by some small, fixed number of random device accesses. This approximation is quite satisfactory for many types of operational systems, but it breaks down in dealing with retrievals from complex file organizations. In a complex information system, an information access (or query) usually requires varying numbers of device accesses, depending on data base size, content, file organization, etc. Clearly, additional techniques are required to deal with performance in these types of data base systems.

There are at least the following three main categories of work on performance: (1) description of system load; (2) simulation of proposed system hardware and software configurations; and (3) synthesis and optimization of system hardware and software configurations.

system load

There are a number of ways of specifying system load. In scientific computation, various instruction mixes, procedural program mixes, or trace tapes have been used. These descriptions are not quite appropriate for the design of information systems. Instruction mixes do not represent the workload on mass storage devices. In the case of procedural programs and trace tapes, each assumes some fixed file structure. Such load specifications preempt any possibility of studying different file organizations for the same problem.

Since file organization is an important consideration in the design of information systems, a system workload description should not contain any commitment to a particular stored file organization. (In other words, it must be data structure independent.) This means that the study of data-structure-independent data models and their associated accessing languages has direct application in the area of load description. However, more load description is needed so that a simulator can calculate the number of records to be retrieved during a particular query. For example, a simulator requires information on the number of instances of a particular type, such as the number and size of fields, the number of entity descriptions, types and numbers of transactions, etc. This information must be added to the data element type information to be found in data model descriptions, e.g., the names of the fields and the names of relationships.

An even more important measure of the utility of a workload description is a human factors one. The workload descriptions collected for the tuning of existing file organizations may be extremely complex, when such information can be collected by the computer without human effort. On the other hand, the workload description for the initial design must be relatively simple because it must be constructed by hand. Rarely can a file designer use a workload description that takes him weeks or months to specify; he might prefer to take his chances with an estimate of the file organization.

In the proceedings of the 1972 Fall Joint Computer Conference, Teichrow[86] published an excellent review of the work directly related to system workload description. He singled out the papers of Young and Kent, Lombardi, Langefors, and the Information Algebra, ADS, PSL, TAG, and SYSTEMATICS systems for detailed comparison. A more recent review was presented by Couger[87] in *Computing Surveys*. In recent years, there seems to

have been little work focused only on this specific area. Perhaps the best method for moving forward is to connect the load description research directly with work on a specific system simulator or optimizer. In fact, there have been definitions of system load descriptions for input to specific design systems. Of these, the studies of the ISDOS group on PSL, the IBM group on FOREM,[88] the Martin Marietta Group on the DIAM simulator,[89] and Cardenas' group at UCLA[90] are particularly noteworthy.

data base system simulation

Until recently, there has been surprisingly little detailed simulation of file organization performance. Most early published work tended to use relatively simple approximation equations. Detailed studies have appeared in the field of scientific computation, but they have related to queuing, paging, and single record random or sequential access. As we mentioned earlier in this paper, such techniques are adequate for studying simple types of transactions directed to simple file organizations, but something more is needed for many systems. There are two possible techniques for achieving the more detailed level of modeling that is necessary, namely, analytical models and event simulation models.

Analytical models generally require a tenth of a second or less to evaluate each information query, independently of the number of device accesses required. When many device accesses are involved in the evaluation, they can be orders of magnitude faster than real time. Their one major drawback is that it is extremely difficult or impossible to describe multiple programs accessing complex files in terms of analytical equations.

Event simulation models, on the other hand, can deal with any amount of detailed interaction. Their problem is that they generally run slower than analytical models. The simulation of a single device access generally requires about one to ten milliseconds. Clearly there is a tradeoff between the two techniques, and each has its area of relevance.

Perhaps the first detailed model of file organization performance was the analytical model FOREM I.[88] This model used analytical equations to calculate timings for file organizations that use secondary indexes, direct, indexed sequential, and sequential access methods on a variety of peripheral devices. The FOREM load description is relatively independent of its description of the stored file organization, so that new organizations may be simulated with a minimal amount of model change. For single thread programs, FOREM achieved approximately the ten percent accuracy that is judged to be reasonable for such models.[91] The FOREM model, however, requires a new program module for each new data structure. The model of Cardenas[90] continued this philosophy, but extended the file organizations covered to multilists and doubly chained trees. A model with a similar philosophy

but using stochastic techniques has been published by Siler.[92] The model of Yao and Merten[93] allowed for a generalized description of the indexing structure for a document retrieval system. This line of analytical modeling leads into one form of file organization optimization. A further improvement on this model was reported by Teorey and Das.[94] This improvement provided a measure of ability to deal with multiaccess environments. An example of a computer science, queuing theory model that has some added data base aspects may be found in Miyamoto.[95]

The difficulty of accurately simulating multiaccess file organization environments in detail with equations also led to a second line of development, based on classical event simulation techniques. An early entry into this area was FOREM Phase II,[96] developed by Owens. A more recent entry is the model of Reiter[97] that incorporates more operating system characteristics. Finally, there is the DIAM-based Martin Marietta model[98] that is mentioned earlier in this paper. A review of the modeling area has been made by Morgan and Kennedy.[99]

synthesis and optimization of data base systems

In spite of a great need and great challenge, only a small amount of work has been done in the past on the automatic selection of information system configurations. This is probably because of the great complexity of the automatic design problem and the difficulty of finding assumptions that make the problem tractable. The early papers of Severance[100] on file organization selection and Shniederman[101] on reorganization points were particularly interesting. More recently, there have been some pioneering efforts by Dearnley and Stocker[102] on self-organizing systems and by Astrahan and Ghosh[31] on optimal search path selection.

Logical design

In many papers, the reader will find the term "logical design" used. If one means logical level in its pure sense, i.e., a description of the real-world associations among entities, it is not possible to invent algorithms to design a logical level. We can understand this by looking at an example. Assume that we wish to describe the real-world relationships between some entities made mainly of wood and some owners of these entities. To describe this situation in a computer, we have to assign unique names to the entities and to the relationships. Assume further that we guess that there is really no significant difference between the wooden entities and the other entities who happen to own them. We might then assign a unique ENTITY NUMBER to each entity, wooden or owner, and name the relationship OWNS. In this design, it would be easy to write a program to list all the entities, but it might take more work to write a program to list

the owners. In the second case, the user would have to write a program that includes a test to determine whether a particular entity number is related to another entity number by the relationship that it is owned by the second entity number.

In a second possible logical design, we might guess that it would be better to place the entities into two different sets for the purpose of naming, say, OWNERS and FURNITURE. In this case, the second program would be easy to write, since it would only have to say LIST OWNERS. This is in contrast to the first one, which would become more difficult, because it would have to say LIST OWNERS and then LIST FURNITURE.

As we can see, the relative efficiency of the two logical designs is measured in terms of the human efficiency of writing the programs to be used in the system. To evaluate this efficiency, we must first have a proposal for the entity sets, and then we must write programs in terms of these sets. Clearly, a computer cannot write the programs, and, if it does not have the programs, it has no way of calculating either the human cost or the computer cost for building and running the system.

Of course, a person could design a logical level and write all the required programs. We might then even invent a way to have the computer calculate an absolute evaluation of the logical level, but we still would not know whether the design were close to optimal. We might also design and program two or more alternatives and have the computer compare them, but that would almost certainly not be worth the trouble. What we should do, and continue to do, is something that people do better than computers; that is, look at the real world and classify its ill-defined elements for our particular ill-defined purposes.

Looking at the design problem in this way, we can see that in the strictest sense a computer cannot do logical design; at present, we do not even have algorithms for the computer-comparison of logical designs. In effect, what a computer does when it executes an algorithm that groups associated fields into records is physical design. Many workers call that logical design because they believe that single-level or hierarchic records are logical structures. Almost without exception, however, design procedures start with given sets of entities and binary associations between them.

Physical design

Taken at its most exact and detailed level, physical design is extremely complex. In an hour, a large system may process thousands of transactions, make millions of peripheral device

accesses, and use billions of computer instructions. The problem of physical design is to find a file organization that is close to optimal for periods of days or months. We clearly cannot solve this problem by simulating each computer instruction for a wide variety of choices of physical file organization. Each instruction-for-instruction simulation of a proposed file organization would run orders of magnitude slower than real time, and it might take months to simulate only one choice. To approach this problem, we make simplifying assumptions and/or localize area of optimization. (For example, the data base simulators previously mentioned all make the assumption that the execution of computer instructions in a transaction can be represented by a fixed time for execution, so that individual instructions need not be simulated. This speeds up the simulation by at least a factor of a thousand.) It is even difficult to make reasonable simplifying assumptions. A change of one percent in a record size or in internal execution time can result in a factor-of-two difference in total processing time. These discontinuities make it virtually impossible to use mathematical optimization techniques in a straightforward way. Much ingenuity must be exercised to find useful equations.

Even without using mathematical techniques, it is possible to make some useful simplifying assumptions. An example of extreme simplification may be found in the paper of Severance and Duhne,[103] entitled "A practitioner's guide to addressing algorithms." Other simplifications are discussed later in the section.

In considering the localization of simplifications, we first discuss the technique of hash addressing. It is possible to look for an optimal design that is relatively independent of other file design considerations. In a series of papers, Lum and coworkers[104] have demonstrated that division by a relatively prime number is, on the average, the best hashing technique, and therefore the best first choice as a technique. In another series of papers, Van der Pool[105] has provided guidance on the selection of an optimal loading factor, considering both storage cost and access time cost. The area of hash addressing techniques has been reviewed by Severance,[106] and more recently by Mauer and Lewis.[107]

Another area of interest is design assistance in grouping associated fields together into records or in deciding whether certain associations should be represented by intersegment hierarchies or DBTG sets. Since a large data base may contain thousands of possible associations, it is extremely useful to have assistance in assuring that all associations are represented and consistent. This kind of assistance is provided by the IBM Data Base Design Aid[108] described in this issue by Raver and Hubbard.[109] Like most of the following design aids, the Raver and Hubbard data base design aid begins with an input of the

required binary associations for the system. From this information, the system designs a network structure to support all the required relations and checks to determine that there are no conflicts. It then also checks to see that hierarchic structures for supporting user logical views can be derived from this structure according to IMS/VS rules for hierarchic records.

A further step has been presented by Smith and Mommens.[110] Here, they ask for weightings of the associations to indicate which associations are traversed most frequently. Their program then performs a pruned exhaustive evaluation of all the possible IMS/VS structures that fulfill the data requirements. The pruning is done on the basis of allowing only valid IMS/VS physical structures and throwing away proposed structures that fall below an already calculated structure in performance. Bubenko et al.[111] require that the user propose valid structures, and they then give an algorithm that uses similar measures to compare structures.

Finally, there are the studies that create equations that can be used with mathematical optimization techniques. Hoffer and Severance[112] use a cluster analysis algorithm to perform allocation of fields to records on the basis of access path traversals. Mitoma and Irani[113] go one step further than previous studies in load description by asking the user to provide a sample of the programs (or run units) to be used against a DBTG data base. There is then a process that goes from these more data-independent descriptions to the providing of traversal frequencies for the proposed paths. The Mitoma-Irani optimization techniques transfer the problem into terms of the shortest path in a network.

There are also a number of studies on the selection and design of indices. Yao and Merten[93] utilize a gradient projection method to design a multilevel index for a document retrieval file. In an earlier paper, Lum and Ling[114] present analytical equations to help in the selection of multiple secondary indices, and Schkolnick[115] presents techniques for a similar problem.

In conclusion, after a period during which little had been done on file design algorithms, there emerged many new and interesting techniques that are currently being studied in research and development. Some of these techniques have already seen use, and we can foresee more successes in the area of automatic design.

Future developments

Recently the National Bureau of Standards and the Association for Computing Machinery held a workshop on *Data Base Directions—The Next Steps*.[116] The workshop had panels on User

Experience, Standardization, Audit, Government Regulations, and Evolving Technology. The panel on Standardization supported immediate standardization based on DBTG. In the panel on Evolving Technology, it became clear that there still exists no broad basis of understanding and agreement to support a particular proposal for standardization.

The panel on Evolving Technology investigated a number of other areas. In system performance, the panel concluded that the critical missing feature is the ability to specify requirements for a specific user's data base system; that is, there is still no good means of describing job load. There did, however, seem to be some hope of developing a formal language and graphics techniques for the specification task. Overall, the panel members speculated that useful job load description products would not be available for the next five years. It also appeared that there would be some improvement in facilities for manual tuning of data base systems, but that again automatic configuration and tuning was a long-term project.

The area of data models has been compared to the early automobile industry, in which many technologies had strong proponents, but some technologies are now seen to have been early portions of the automotive learning curve. In such a situation, it is not possible to predict the success of any given technology, especially one that is in a very early portion of its learning curve.

It was expected that approximately five years would be required to determine whether relational systems have significant advantages over existing technology, and another three years to settle the same issues for binary and set-oriented systems. With this in mind, it was the consensus that—for systems in planning—it would be best for users to depend on existing technology.

Attention was also paid to the use of more semantic information and the use of inferential techniques in data base systems. The general opinion was that there would be a gradual progression of the use of these techniques. Simple techniques (like virtual fields and representation mode discrimination) are already being used. However, it was expected that several years of work would be required before systems that utilized long chains of reasoning would become commercially useful.

Concluding remarks

In summary, there are two efficiencies of importance to the development of data base systems. No matter how inexpensive the hardware becomes, the increasing size and speed of data base

systems will require high computer efficiency. The new force in the field, however, is the increasing demand for data base systems that are efficient in the utilization of human resources, users, programmers and systems analysts. Considerable research and development is going on, and this work should contribute significantly to the evolution of easier-to-use, more powerful data base systems.

CITED REFERENCES

1. M. E. Senko, E. B. Altman, M. M. Astrahan, and P. L. Fehder, "Data structures and accessing in data-base systems," *IBM Systems Journal* **12,** No. 1, 30–93 (1973).
2. B. W. Boehm, "Software and its impact: a quantitative assessment," *Datamation* **19,** No. 5, 48–59 (1973).
3. J. P. Fry and E. H. Sibley, "Evolution of data-base management systems," *ACM Computing Surveys* **8,** No. 1, 7–42 (1976).
4. L. J. Cohen, *Data Base Management Systems*, Q.E.D. Information Sciences, Inc., Wellesley, MA (1976).
5. *Datapro 70*, Datapro Research Corporation, Delran, NJ (1976).
6. *Auerbach Computer Technology Reports: Segment J*, Auerbach Publishers, Inc., Philadelphia, PA (1976).
7. R. E. Blier, "Treating hierarchical data structures in the SDC time shared data management system (TDMS)," *Proceedings of the ACM 22nd National Conference*, p. 67, 41–49, Thompson Book Co., Washington, DC (1967).
8. W. C. McGee, "The IMS/VS system," *IBM Systems Journal* **16,** No. 2, 84–168 (1977).
9. *Interactive Query Facility (IQF), General Information Manual*, GH20-1074, IBM Corporation, Data Processing Division, White Plains, New York 10604.
10. *Generalized Information System, General Information Manual*, GH20-9035, IBM Corporation, Data Processing Division, White Plains, New York 10604.
11. J. H. Wimbrow, "A large-scale interactive administrative system," *IBM Systems Journal* **10,** No. 4, 260–282 (1971).
12. J. E. Siwiec, "A high-performance DB/DC system," *IBM Systems Journal* **16,** No. 2, 169–195 (1977).
13. D. C. Tsichritzis and F. H. Lochovsky, "Hierarchical data-base management," *ACM Computing Surveys* **8,** No. 1, 105–124 (1976).
14. B. C. M. Douque and G. M. Nijssen, "The Wepion recommendations on the CODASYL DDL 1973," *Data Base Description*, North-Holland Publishing Co., Amsterdam (1975), pp. 369–372.
15. T. B. Steel, Jr., "Summary of recommendations," *Data Base Description*, North-Holland Publishing Co., Amsterdam (1975), pp. 373–376.
16. F. Manola, *The CODASYL data description language: status and activities, April 1975*, NRL Report 8038, Naval Research Laboratory, Washington, DC (1976).
17. R. W. Taylor and R. L. Frank, "CODASYL data-base management systems," *ACM Computing Surveys* **8,** No. 1, 67–104 (1976).
18. G. F. Duffy and F. P. Gartner, "An on-line information system for management," *AFIPS Conference Proceedings*, Spring Joint Computer Conference **34,** 339–350 (1969).
19. *Interactive Query and Report Processor, General Information Manual*, GB21-9903, IBM Corporation, Data Processing Division, White Plains, New York 10604.
20. CODASYL Data Base Task Group, *April 1971 Report*, ACM, New York, NY (1971).

21. ISO/TC 97/SC 5 Study Group on Data Base Management Systems, *Conclusions of the June 24–26 Meeting*, Washington, DC (1975).
22. ANSI/X3/SPARC Study Group on Data Base Management Systems, "Interim report," *FDT Bulletin of ACM SIGMOD* **7**, No. 21, 140 (1975).
23. GUIDE-SHARE Database Requirements Group, *Database Management System Requirements*, SHARE Inc., New York, NY (1970).
24. T. B. Steel, Jr., "Formalization of conceptual schemas," *Proceedings of the PL/I Symposium, Keystone, CO, February 1976*, CIBAR Corporation, Colorado Springs, CO, 1976; also to appear in T. B. Steel, Jr. and J. M. Gallitano, *A Proposal of a Language for the Conceptual Schema*, North-Holland Publishing Co., Amsterdam (1977); to be published.
25. S. E. Madnick and J. W. Alsop, "A modular approach to file system design," *AFIPS Conference Proceedings*, Spring Joint Computer Conference **34**, 1–13 (1969).
26. H. S. Meltzer, *Data Base Concepts and Architecture for Data Systems*, IBM Report to SHARE Information Systems Research Project, SHARE Inc., New York, NY (August 20, 1969).
27. S. McIntosh and D. Griffel, "ADMINS from Mark III to Mark V," *Proceedings of the IFIPS Congress* **68**, North-Holland Publishing Co., Amsterdam (1969).
28. C. T. Davies, *A Logical Concept for Control and Management of Data*, AR-0803-00, IBM Corporation, System Development Division, Poughkeepsie, New York (1967).
29. E. F. Codd, "A relational model for large shared data banks," *Communications of the ACM* **13**, No. 6, 377–387 (1970).
30. P. L. Fehder, *The Representation-Independent Language. Part 1: Introduction and Subsetting Operations*, Research Report RJ 1121, IBM Research Laboratory, San Jose, California (1972).
31. S. P. Ghosh and M. M. Astrahan, "A translator optimizer for obtaining answers to entity set queries from an arbitrary access path network," *Information Processing 71*, North-Holland Publishing Co., Amsterdam (1974), pp. 436–439. S. P. Ghosh and M. E. Senko, "On the analysis of search path procedures in an information system based on string paths," *IBM Journal of Research and Development* **18**, 408–422 (1974).
32. P. P-S. Chen, "The entity-relationship model—toward a unified view of data," *ACM Transactions on Database Systems* **1**, No. 1, 9–36 (1976).
33. G. M. Held, M. R. Stonebraker, and E. Wong, "INGRES—a relational data base system," *Proceedings of the AFIPS National Computer Conference* **44**, 409–416 (1975).
34. J. Mylopoulos, S. Schuster, and D. Tsichritzis, "A multi-level relational system," *Proceedings of the AFIPS National Computer Conference* **44**, 403–408 (1975).
35. V. K. M. Whitney, "RDMS: a relational data management system," *Proceedings of the Fourth International Symposium on Computer and Information Sciences (COINS IV)* (1972).
36. D. McLeod and M. Meldman, "RISS—A generalized minicomputer relational data base management system," *Proceedings of the AFIPS National Computer Conference* **44**, 397–402 (1975).
37. L. S. Schneider, "A relational view of the Data Independent Accessing Model," *Proceedings of the ACM-SIGMOD International Conference on Management of Data* (1976), pp. 75–90.
38. H. A. Schmid and P. A. Bernstein, "A multi-level architecture for relational data base systems," *Proceedings of the International Conference on Very Large Data Bases, Framingham, MA, 1975*, D. S. Kerr, editor, ACM, New York, NY, 1975, pp. 202–226.
39. G. K. Manacher, "On the feasibility of implementing a large relational data base with optimal performance on a mini-computer," *Proceedings of the International Conference on Very Large Data Bases, Framingham, MA, 1975*, D. S. Kerr, editor, ACM, New York, NY, 1975, pp. 175–201.

40. L. S. Schneider, Martin Marietta Corp., Denver, CO; private communication (1976).
41. C. J. Date, *An Introduction to Database Systems, 70,* Addison-Wesley Publishing Co., Reading, MA (1975).
42. M. M. Zloof, "Query-by-Example," *Proceedings of the AFIPS National Computer Conference* **44,** 431–438 (1975).
43. J. C. Thomas and J. D. Gould, "A psychological study of Query-by-Example," *Proceedings of the AFIPS Computer Conference* **44,** 439–445 (1975).
44. W. Kent, "New criteria for the conceptual model," *Systems for Large Data Bases, Preprints of the Conference on Very Large Data Bases, Brussels, Belgium, September, 1976,* P. C. Lockemann and E. J. Neuhold, editors, North-Holland Publishing Co., Amsterdam (1976). W. Kent, "Entities and relationships in information," *Modeling in Data Base Management Systems, Proceedings of the IFIP-TC-2 Working Conference, Nice, France, January, 1977,* IRIA, 78150 LeChesnay, France (1977).
45. G. Bracchi, P. Paolini, and G. Pelagatti, "Binary logical associations in data modeling," *Modeling in Data Base Management Systems, Proceedings of the IFIP-TC-2 Working Conference, Freudenstadt, Germany, January, 1976,* G. M. Nijssen, editor, North-Holland Publishing Co., Amsterdam (1976), pp. 125–148.
46. H. A. Schmid and J. R. Swenson, "On the semantics of the relational model," *Proceedings of the SIGMOD Conference, San Jose, CA, 1975,* ACM, New York, NY (1975), pp. 211–223.
47. W. A. Woods and R. M. Kaplan, *The lunar sciences natural language information system,* BBN Report 2265, Bolt Beranek and Newmann, Inc., Cambridge, MA (1971).
48. W. J. Plath, "REQUEST: a natural language question-answering system," *IBM Journal of Research and Development* **20,** No. 4, 326–335 (1976).
49. S. R. Petrick, "On natural language based computer systems," *IBM Journal of Research and Development* **20,** No. 4, 314–325 (1976).
50. R. E. Levien and M. E. Maron, "A computer system for inference execution and data retrieval," *Communications of the ACM* **10,** 715–721 (1967).
51. W. L. Ash and E. H. Sibley, "TRAMP: an interpretive associative processor with deductive capabilities," *Proceedings of the ACM 23rd National Conference,* Brandon/Systems Press, Princeton, NJ (1968), pp. 144–156.
52. J. A. Feldman and P. D. Rovner, "An ALGOL-based associative language," *Communications of the ACM* **12,** No. 8, 439–449 (1969).
53. M. E. Senko, "DIAM as a detailed example of the ANSI SPARC architecture," *Modeling in Data Base Management Systems, Proceedings of the IFIP-TC-2 Working Conference, Freudenstadt, Germany, January, 1976,* G. M. Nijssen, editor, North-Holland Publishing Co., Amsterdam (1976), pp. 73–94.
54. B. Langefors, "Information systems," *Information Processing 74,* North-Holland Publishing Co., Amsterdam (1974), pp. 937–945.
55. B. Sundgren, *An Infological Approach to Data Bases, (Urval nr 7),* National Central Bureau of Statistics, Stockholm, Sweden (1973).
56. P. Titman, "An experimental data base system using binary relations," *Data Base Management, Proceedings of the IFIP-TC-2 Working Conference, Cargese, Corsica, January, 1974,* J. W. Klimbie and K. L. Koffeman, editors, North-Holland Publishing Co., Amsterdam (1974).
57. G. Bracchi, A. Fedeli, and P. Paolini, "A multilevel relational model for data base management systems," *Data Base Management, Proceedings of the IFIP-TC-2 Working Conference, Cargese, Corsica, January, 1974,* J. W. Klimbie and K. L. Koffeman, editors, North-Holland Publishing Co., Amsterdam (1974).
58. J-R. Abrial, "Data semantics," *Data Base Management, Proceedings of the IFIP-TC-2 Working Conference, Cargese, Corsica, January, 1975,* J. W. Klimbie and K. L. Koffeman, editors, North-Holland Publishing Co., Amsterdam (1974).

59. M. E. Senko, "The DDL in the context of a multilevel structured description: DIAM II with FORAL," *Data Base Description, Proceedings of the IFIP-TC-2 Working Conference, Wepion, Belgium, January, 1975*, B. C. M. Douque and G. M. Nijssen, editors, North-Holland Publishing Co., Amsterdam (1975), pp. 239–258.
60. J. A. Bubenko and S. Berild, *CADIS System 4: a Tool Incremental Description and Analysis of Systems*, Report TRITA-IBADB-3082, Department of Information Processing, University of Stockholm, Sweden (1974).
61. S. Berild and S. Nachmens, "Some practical application of CS4—a DBMS for associative data bases," *Preprints, Proceedings of the IFIP-TC-2 Working Conference, Nice, France, January, 1977, Modeling in Data Base Management Systems*, IRIA, 78150 Le Chesnay, France (1977).
62. E. Falkenberg, "Concepts for modeling information," *Modeling in Data Base Management Systems, Proceedings of the IFIP-TC-2 Working Conference, Freudenstadt, Germany, January, 1976*, G. M. Nijssen, editor, North-Holland Publishing Co., Amsterdam (1976), pp. 95–110.
63. P. Hall, J. Owlett, and S. Todd, "Relations and entities," *Modeling in Data Base Management Systems, Proceedings of the IFIP-TC-2 Working Conference, Freudenstadt, Germany, January, 1976*, G. M. Nijssen, editor, North-Holland Publishing Co., Amsterdam (1976), pp. 201–220.
64. E. Falkenberg, "Design and application of a natural language oriented data base language," *Advanced Course on Data Base Languages and Natural Language Processing*, H. J. Schneider, editor, Technical University, Berlin, Germany (1975).
65. J. A. Bubenko, "The temporal dimension in information modeling," *Modeling in Data Base Management Systems, Proceedings of the IFIP-TC-2 Working Conference, Nice, France, January, 1977*, IRIA, 78150 Le Chesnay, France (1977) pp. 41–66.
66. B. M. Schueler, "Update Reconsidered," *Modeling in Data Base Management Systems, Proceedings of the IFIP-TC Working Conference, Nice, France, January, 1977*, IRIA 78150 Le Chesnay, France (1977).
67. L. A. Bjork, Jr. "Generalized audit trail requirements and concepts for data base applications," *IBM Systems Journal* **14,** No. 3, 229–245 (1975).
68. L. Kerschberg, A. Klug, and D. Tsichritzis, "A taxonomy of data models," *Systems for Very Large Data Bases, Preprints of the Conference on Very Large Data Bases, Brussels, Belgium, 1976*, P. C. Lockemann and E. J. Neuhold, editors, North-Holland Publishing Co., Amsterdam (1976), pp. 43–64.
69. M. E. Senko, "Conceptual schemas, abstract data structures, enterprise descriptions," *Proceedings of ICS77, the ACM International Computing Symposium 1977, Liege, Belgium*, North-Holland Publishing Co., Amsterdam (1977).
70. M. E. Senko, "DIAM II with FORAL LP: making pointed queries with light pen," *Proceedings of the IFIP Congress 77, Toronto, Canada, 1977. FORAL LP for DIAM II: FORAL with light pen—a language primer*, Research Report RC-6328, IBM Thomas J. Watson Research Center, Yorktown Heights, NY (1976).
71. M. E. Senko, "FORAL II for DIAM II, information structure and query-maintenance language," may be obtained from the author, IBM Thomas J. Watson Research Center, Yorktown Heights, NY (1976).
72. D. P. Smith, *An Approach to Data Description and Conversion*, Moore School Report No. 72-20, Moore School of Electrical Engineering, University of Pennsylvania, Philadelphia, PA (1971).
73. R. W. Taylor, *Generalized Data Base Management System Data Structures and their Mapping to Physical Storage*, Ph.D. Thesis, University of Michigan, Ann Arbor, MI (1971). E. H. Sibley and R. W. Taylor, "A data definition and mapping language," *ACM Communications* **16,** No. 12, 750–759 (1973).
74. The Stored-Data Definition and Translation Task Group of the CODASYL Systems Committee, "Stored-data description and data translation: a model

and a language," *Information Systems* **2,** No. 3 (1977).
75. E. Nahouraii, L. O. Brooks, and A. F. Cardenas, "An approach to data communication between different generalized data base systems," *Systems for Large Data Bases, Proceedings of the Conference on Very Large Data Bases, Brussels, Belgium, September, 1976,* North-Holland Publishing Co., Amsterdam (1976).
76. M. E. Senko, "DIAM II and levels of abstraction: the physical device level: a general model for access methods," *Systems for Large Data Bases, Proceedings of the Conference on Very Large Data Bases, Brussels, Belgium, September, 1976,* North-Holland Publishing Co., Amsterdam (1976).
77. A. Cabanes, "Data independence and physical implementation," *Data Structure Models for Information Systems, Travaux de l'Institut d'Informatique No. 4, Proceedings of the International Workshop, Namur, Belgium, May, 1974,* Presses Universitaires de Namur, Namur, Belgium (1975), pp. 169–188.
78. M. Adiba and C. Delobel, "The problem of the cooperation between different DBMS," *Modeling in Data Base Management Systems, Proceedings of the IFIP-TC-2 Working Conference, Nice, France, January, 1977,* IRIA, 78150 Le Chesnay, France (1977), pp. 131–156.
79. F. L. Hainaut, "Some tools for data independence in multilevel data base systems," *Modeling in Data Base Management Systems, Proceedings of the IFIP-TC-2 Working Conference, Nice, France, January, 1977,* IRIA, 78150 Le Chesnay, France (1977), pp. 157–188.
80. J. F. Kelly, *Computerized Management Information Systems,* The Macmillan Co., New York, NY (1970), p. 533.
81. *DB/DC Data Dictionary, General Information Manual,* GH20-9104-0, IBM Corporation, Data Processing Division, White Plains, New York 10604.
82. P. P. Uhrowczik, "Data dictionaries/directories," *IBM Systems Journal* **12,** No. 4, 332–350 (1973).
83. R. G. Canning, "The data dictionary/directory function," *EDP Analyzer* **12,** No. 11 (1974).
84. P. H. Seaman and R. C. Soucy, "Simulating operating systems," *IBM Systems Journal* **8,** No. 4, 264–279 (1969).
85. P. H. Seaman, R. A. Lind, and T. L. Wilson, "On teleprocessing system design, Part IV. An analysis of auxiliary-storage activity," *IBM Systems Journal* **5,** No. 3, 158–170 (1966).
86. D. Teichroew, "A survey of languages for stating requirements for computer-based information systems," *AFIPS Conference Proceedings,* Fall Joint Computer Conference Part II **41,** 1203–1244, (1972).
87. J. D. Couger, "Evolution of business system analysis techniques," *ACM Computing Surveys* **5,** No. 3, 167–198 (1973).
88. M. E. Senko, V. Y. Lum, and P. J. Owens, "A file organization model (FOREM)," *Information Processing 68, Proceedings of the IFIP Conference,* North-Holland Publishing Co., Amsterdam, (1969), pp. 514–519. M. E. Senko, P. J. Owens, and V. Y. Lum, "File structure simulation model (FSSM)," *Formatted File Organization Techniques, Final Contract Report, Contract (AF 30(602)–4088), Rome Air Development Center,* IBM Thomas J. Watson Research Center, Yorktown Heights, NY (1967).
89. L. S. Schneider and C. R. Spath, "Quantitative data description," *Proceedings of the SIGMOD Conference, San Jose, California, 1975,* ACM, New York, NY (1975), pp. 167–185.
90. A. F. Cardenas, "Evaluation and selection of file organization—a model and a system," *ACM Communications* **16,** No. 9, 540–548 (1973). A. F. Cardenas and J. P. Sagamag, "Modeling and analysis of data base organization, the doubly chained tree structure," *Information Systems* **1,** 57–67 (1975).
91. V. Y. Lum, M. E. Senko, H. Ling, and J. H. Barlow, "Quantitative timing analysis and verification for file organization modeling," *Information Sys-*

tems, COINS IV, J. Tou, editor, Plenum Press, New York, NY (1974), pp. 377–386.
92. K. F. Siler, "A stochastic evaluation model for database organizations in data retrieval systems," *ACM Communications* **19,** No. 2, 84–95 (1976).
93. S. B. Yao and A. G. Merten, "Selection of file organization using an analytical model," *Proceedings of the International Conference on Very Large Data Bases, Framingham, MA, September, 1975*, D. S. Kerr, editor, ACM, New York, NY (1975), pp. 255–267.
94. T. J. Teory and K. S. Das, "Application of an analytical model to evaluate storage structures," *Proceedings of the 1976 SIGMOD Conference on Management of Data, Washington, DC, June, 1976*, J. B. Rothnie, editor, ACM, New York, NY (1976), pp. 9–20.
95. I. Miyamoto, "Hierarchical performance analysis models for data base systems," *Proceedings of the International Conference on Very Large Data Bases, Framingham, MA, September, 1975*, D. S. Kerr, editor, ACM, New York, NY (1975), pp. 322–352.
96. P. J. Owens, "Phase II—a data base management modeling system," *Information Processing 71, Proceedings of the IFIPS 1971 International Conference*, North-Holland Publishing Co., Amsterdam (1972), pp. 827–832.
97. A. Reiter, "Data models for secondary storage representations," *Proceedings of the International Conference on Very Large Data Bases, Framingham, MA, September, 1975*, D. S. Kerr, editor, ACM, New York, NY (1975), pp. 87–119.
98. L. S. Schneider, *Generalized Data Management System, Math Model Simulator, User Guide, NASA Contract NAS9-13951*, Institutional Data Systems Division, Lyndon B. Johnson Space Center, NASA, Houston, TX (1975).
99. H. L. Morgan and S. R. Kennedy, "Mathematical models of file processing: survey and classification," *Information Science Technical Report No. 3*, California Institute of Technology, Pasadena, CA (1972).
100. D. G. Severance and A. G. Merten, "Performance evaluation of file organization through modeling," *Proceedings of the ACM National Conference, 1972*, ACM, New York, NY (1972), pp. 1061–1072.
101. B. Schneiderman, "Optimum data base reorganization points," *ACM Communications* **16,** 362–365 (1973).
102. P. M. Stocker and P. A. Dearnley, "A self-organizing data base management system," *Data Base Management, Proceedings of the IFIP TC-2 Working Conference, Cargese, Corsica, April, 1974*, J. W. Klimbie and K. L. Koffeman, editors, North-Holland Publishing Co., Amsterdam (1974), pp. 337–350. P. Dearnley, "A model of a self-organizing data management system," *Computer Journal* **17,** 13–16 (1974).
103. D. Severance and R. Duhne, "A practitioner's guide to addressing algorithms," *ACM Communications* **19,** No. 6, 314–326 (1976).
104. V. Y. Lum, P. S. T. Yuen, and M. Dodd, "Key-to-address transform techniques: a fundamental performance study on large existing formatted files," *ACM Communications* **14,** No. 4, 228–239 (1971). V. Y. Lum, "General performance analysis of key-to-address transformation methods using an abstract file concept," *ACM Communications* **16,** No. 10, 603–612 (1973). S. P. Ghosh and V. Y. Lum, "An analysis of collisions when hashing by division," *Information Systems* **1,** No. 1, 15–22 (1975).
105. J. A. Van der Pool, "Optimum storage allocation for initial loading of a file," *IBM Journal of Research and Development* **16,** No. 6, 579–586 (1972).
106. D. G. Severance, "Identifier search mechanisms: a survey and a generalized model," *ACM Computing Surveys* **6,** No. 3, 175–194 (1974).
107. W. D. Mauer and T. G. Lewis, "Hash table methods," *ACM Computing Surveys* **7,** No. 1, 5–19 (1975).

108. *Data Base Design Aid. General Information Manual*, GH20-1626, IBM Corporation, Data Processing Division, White Plains, New York, NY 10604.
109. N. Raver and G. Hubbard, "Automated logical file design: concepts and application," this issue.
110. S. E. Smith and J. H. Mommens, "Automatic generation of physical data base structures," *ACM SIGMOD International Conference, San Jose, CA, 1975*, ACM, New York, NY (1975).
111. J. A. Bubenko, Jr., S. Berild, E. Lindencrona-Ohlin, and S. Nachmens, "From information structures to DBTG data structures," *Proceedings of the Conference on Data: Abstraction, Definition, and Structure, FDT Bulletin* **8**, No. 2, 73–85 (1976).
112. J. A. Hoffer and D. G. Severance, "The use of cluster analysis in physical data base design," *Proceedings of the International Conference on Very Large Data Bases, Framingham, MA, September, 1975*, ACM, New York, NY (1975), pp. 69–86.
113. M. F. Mitoma and K. B. Irani, "Automatic data base schema design and optimization," *Proceedings of the International Conference on Very Large Data Bases, Framingham, MA, September, 1975*, ACM, New York, NY (1975), pp. 286–321.
114. V. Y. Lum and H. Ling, "An optimization problem in the selection of secondary keys," *Proceedings of the 1971 ACM National Conference* **26**, 349–356, (1972).
115. M. Schkolnick, "Secondary index optimization," *ACM SIGMOD 1975, International Conference on Management of Data, San Jose, CA*, ACM, New York, NY (1975).
116. *Data Base Directions, The Next Steps, NBS Special Publication 451*, John L. Berg, editor, U.S. Department of Commerce, National Bureau of Standards, U.S. Government Printing Office, Washington, DC (1975).

GENERAL REFERENCES

1. A. Blaser and H. Schmutz, *Data base research: a survey*, Research Report TR75.10.009, IBM Heidelberg Scientific Center, Heidelberg, Germany (1975).

2. C. J. Date, *An Introduction to Database Systems*, Addison-Wesley Publishing Co., Reading, MA (1975).

3. J. Martin, *Computer Data Base Organization*, Prentice-Hall, Inc., Englewood Cliffs, NJ (1975).

Reprint Order No. G321-5053.