Basic concepts of relational data base management systems are described. Characteristics of the relational approach are identified and compared with present implementations of hierarchical and network data base systems. Depending on the application, a user may experience one or more of the following benefits of relational systems described in this paper: ease of understanding, increased data independence, ease of use, sound theoretical basis, and generalized data definition. Types of applications most suited to hierarchical and network data base systems are also compared and contrasted.

A primer on relational data base concepts

by G. Sandberg

For about a decade there has been continuously increasing interest in relational data base systems, most of it initially created by university and research activities. A series of papers published by E. F. Codd in the early seventies are often cited as the earliest works on the subject.[1]

Relational data base systems are now becoming available for operational data processing installations. Examples of such systems by IBM are Query-By-Example[2,3] and IMPS.[4] In addition, many research prototype systems have been implemented, as exemplified by System R[5] and IS/1-PRTV.[6,7] A number of other relational systems are expected to become commercially available within the next few years.[8,9] This paper describes basic concepts of relational data base systems and identifies potential benefits of the relational data base approach, comparing it with present implementations of hierarchic and network data base systems.

What is a relational data base system?

The most fundamental property of a relational data base system is that data are presented to the user as tables instead of networks or hierarchies. Thus, the data are structured in the form of *tables*

Table 1 Relational structure of employee records

		COLUMN OR FIELD		
		1 *EMPLOYEE NUMBER*	2 *NAME*	3 *DEPARTMENT*
ROW OR RECORD	1	61256	MYGIND	NFSC
	2	38972	CHEMNITZ	NMC
	3	09181	BARCLAY	NFSC
	4	74245	SANDBERG	NFSC
	5	22318	PERSSON	NMC

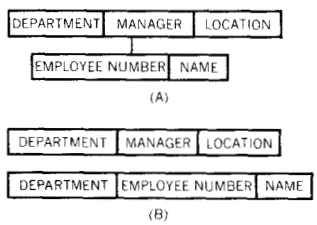
consisting of *columns* and *rows*, with the rows corresponding to records or segments, and the columns representing fields within the records. Table 1 is an example of a relational data structure for employee information, with Employee as the table name.

This illustrative table contains only five rows or records, one for each employee. Such a data base for a company of a size that requires a data base, of course, contains many more rows. Three facts are recorded here for each employee—employee number, name, and department—each in a separate column of the table.

The internal data storage format is not relevant to the relational view. This is not to say that internal access and storage techniques are not important because they determine whether the data base system performance is acceptable. Performance implications, however, are not part of the definition of the relational view.

The important fact is that the relational view exists at the level at which the user sees the data. The user may be, for example, a person sitting at a visual terminal and interacting with the system in a specialized query language or a programmer using conventional programming languages like COBOL or PL/I.

Figure 1 Transformation of (a) a hierarchy of two record types into (b) the corresponding relational tables



Any hierarchical or network data structure can be transformed into a set of relational tables. One technique is to convert each predefined access path in the network or hierarchy into a key field column in a relational table. Then all fields from the hierarchy or network record are explicitly named in the relational table. As an example: two tables may be substituted for a parent-child record structure in a hierarchy or an owner-member set in a CODASYL network. The first table represents the parent record type, and the second is equivalent to the child record type, expanded with the key field of the parent as an extra column. The transformation of a hierarchy of two record types (a) into corresponding relational tables (b) is illustrated in Figure 1.

table definition

If a relational data base view is simply a view of records with the same format, how does that differ from program views of traditional flat files that have been used for many years? The differences include specificity of rules. The following are rules that must be followed if the data base view is to qualify as a relational view:

- Each table contains only one record type.
- Each record (row) has a fixed number of fields, all of which are explicitly named.
- Fields are distinct (atomic) so that repeating groups are not allowed.[10]
- Each record is unique—duplicates are not allowed.
- Records may come in any order; there is no predetermined sequence.
- Fields take their values from a *domain* of possible field values.
- The same domain may be used for many different field types, thus becoming the source of field values in different columns in the same or different tables.
- New tables can be produced on the basis of a match of field values from the same domain in two existing tables.

The formation of new tables is a key to relational systems, and does not apply to access methods handling flat files. The access methods are not designed to combine such files into new files; that is an application program responsibility.

Table operations

One of the new operations available in relational systems is the capability of combining relational tables, called a *join*. Other relational operations are *selection* (which creates a subset of all the records in a table), and *projection* (which creates a subset of the columns in a table). A key characteristic shared by all relational operations is that the results they produce are always new tables. This makes it possible to provide very powerful and concise languages for the manipulation of relational data structures.

selection

The simplest of these basic relational operations is selection, in which certain rows in a given table are selected and used to build a new table. A selection criterion may be, for example, that one or more fields have a specific value: all rows satisfying this condition are selected for the new table. Table 2 gives an example of selection. Here all rows of Table 2a in which the employee's department is NFSC are selected for inclusion in the newly created Table 2b.

projection

In the next operation—projection—only certain columns in a given table are selected and used to build a new table with fewer

Table 2 Selection of employees in (a) who are in Department NFSC for inclusion in a new table (b)

(a)

EMPLOYEE NUMBER	*NAME*	*DEPARTMENT*
61256	MYGIND	NFSC
38972	CHEMNITZ	NMC
09181	BARCLAY	NFSC
74245	SANDBERG	NFSC
22318	PERSSON	NMC

(b)

EMPLOYEE NUMBER	*NAME*	*DEPARTMENT*
61256	MYGIND	NFSC
09181	BARCLAY	NFSC
74245	SANDBERG	NFSC

Table 3 Projection using the NAME and DEPARTMENT column in (a) to form the new table (b)

(a)

EMPLOYEE NUMBER	*NAME*	*DEPARTMENT*
61256	MYGIND	NFSC
38972	CHEMNITZ	NMC
09181	BARCLAY	NFSC
74245	SANDBERG	NFSC
22318	PERSSON	NMC

(b)

NAME	*DEPARTMENT*
MYGIND	NFSC
CHEMNITZ	NMC
BARCLAY	NFSC
SANDBERG	NFSC
PERSSON	NMC

columns. When the new table is built, the resulting table may contain some rows that are identical, because in some rows values in the retained columns may be identical. Since duplicates are not allowed in a relational table, all but one of such duplicate rows are discarded. The basic operation is illustrated in Table 3, in which the projection operation is performed on the Name and Department columns in Table 3a to form the resultant Table 3b. There are no duplicate rows in this example.

Table 4 Join on DEPARTMENT (c) consists of MANAGER and LOCATION information in (b) combined with EMPLOYEE NUMBER and NAME in (a)

(a)

EMPLOYEE NUMBER	*NAME*	*DEPARTMENT*
61256	MYGIND	NFSC
38972	CHEMNITZ	NMC
09181	BARCLAY	NFSC
74245	SANDBERG	NFSC
22318	PERSSON	NMC

(b)

DEPARTMENT	*MANAGER*	*LOCATION*
NFSC	JARENO	STOCKHOLM
NMC	HOFFMAN	COPENHAGEN

(c)

EMPLOYEE NUMBER	*NAME*	*DEPARTMENT*	*MANAGER*	*LOCATION*
61256	MYGIND	NFSC	JARENO	STOCKHOLM
38972	CHEMNITZ	NMC	HOFFMAN	COPENHAGEN
09181	BARCLAY	NFSC	JARENO	STOCKHOLM
74245	SANDBERG	NFSC	JARENO	STOCKHOLM
22318	PERSSON	NMC	HOFFMAN	COPENHAGEN

Quite often, selection and projection are combined into the same request. In that case, the search criterion may be that a certain field be greater than a specified value, and only certain named columns are of interest. The first operation selects the rows that satisfy the size condition, and the second operation projects the relevant columns.

join

The third operation—join—means that two tables are to be merged on the basis of the values from one column in each table.[11] The two tables are said to be joined over the two columns. Consider the example in which Table 4a and Table 4b are required to be joined on the basis of the Department column in each table. When this is done, the Manager and Location columns form the join on Department shown in Table 4c.

Conceptually, the join operation works as follows:

- Take the first row from the first table and try to find a row in the second table with a matching value.
- When a match is found, put the two rows together, forming one new row.
- Continue until the second table is exhausted.
- Take the next row from the first table and search the second table for a match again.

- Repeat until the first table is also exhausted. The second table has now been searched as many times as there are rows in the first table.

For the join operation to make sense, the two columns must contain field values that are comparable, that is, they come from the same domain. To illustrate this point, if there is one domain of all possible dates, and one domain of all possible prices, it is not reasonable to join two tables on the basis of dates in one table and prices in the other. Relational implementations do not always check such conditions, but leave the user to determine what is a reasonable operation.

The method of operation for a join is very time-consuming and expensive if implemented directly as described. That has been a criticism of relational systems since the beginning. However, improved techniques in the areas of query optimization and indexing are developing, some of which are discussed in References 12–14. Thus, in the join operation previously discussed, if there were an index on a column in the second table, only the index might have to be searched. And for some rows in the first table, no search would be required in the second table at all. Further, if there were also an index on a column in the first table, the search for equal values could be performed entirely in the indexes. The data base system may also keep statistics about actual or intended usage, in order to optimize the search order internally. It now seems that improved optimization methods are sufficiently developed to make possible large-scale relational testing.

Access paths

Records can be accessed in a relational data base system only through the matching of field values. There is no path-following mechanism in a relational system that is comparable, for instance, to a FIND LAST WITHIN 'set' operation in a CODASYL system or GET NEXT WITHIN PARENT in DL/I.

In the hierarchic or network approach, access paths are predefined in the data structure seen by the user. A programmer of a hierarchic data structure uses the implicit hierarchic structure to navigate through, for instance, an access path from a parent to a child segment type. However, any new access requirement that does not directly follow the predefined access paths in the data structure requires additional programming logic.

In the relational approach, no paths are predefined in the data structure as seen by the user. Because all access is accomplished by the matching of field values, many different paths potentially exist. This means that the relational approach has considerable

potential for extensions and restructuring, and provides a very high-level interface to the data structures, as compared with data models that use predefined paths.

At the same time, there is increased risk of inefficient and costly data access. Since the user does not see which access paths are internally favored over others, he cannot decide whether optimum paths are being followed.

An interesting and important characteristic of the relational approach is symmetry in data access for all types of access. This results from the equality of fields in the relational data structure. Access in network and hierarchical implementations often requires different coding techniques, depending on the predefined path being followed. Consider, for example, access to a dependent segment in a DL/I hierarchy. This requires different coding statements, depending on whether a predefined secondary index access path or an access path implicit in the hierarchical structure is used. The same applies to access VIA 'set' as opposed to direct access in a CODASYL network. Therefore, such data structures cannot be easily restructured without some effect on existing programs and procedures.[15]

The essentials of relational operations and access paths may be summarized as follows:

- Relational operations work on whole tables, i.e., sets of records.
- The result of each operation is a new table.
- Operations are based on field values in the tables as the one and only means of access.

The characteristics of today's network and hierarchical data structures are the following:

- Network and hierarchical systems operate on individual records, one at a time. This, however, is not an inherent necessity, since set operations on networks and hierarchies are also conceivable.[16]
- The result of data access operations is normally a single record,[17] since network and hierarchical systems work with individual records.
- Operations are based mostly on predefined access paths in the data structures and different access paths may require different coding techniques.

Relational languages

We have discussed the basic relational operations of selection, projection, and join, and now consider a very important aspect of

relational data base systems: how these functions are provided to the user through relational language facilities. Even the form of the language is very important for the ease-of-use aspect of the relational approach.

Many different languages have been defined for use with relational data base structures. Most such languages are query-type languages,[18,19] but there are also languages of the traditional type intended to be incorporated into such programming languages as COBOL and PL/I.[16]

algebraic languages

A language that explicitly provides select, project, and join is called a *relational algebraic language*. An example of an algebraic relational language is SQL,[20] which, by the way, is the language used in System R.[21] Algebraic relational languages work with sets of records, that is, they work on tables as a whole. Other operations that work on sets of records are the classical set operations from mathematical set theory: union, intersection, and difference. These operators are sometimes included in relational algebraic languages, as well as special functions for summation, aggregation, and ordering.

calculus languages

Another class of relational languages is that of the *relational calculus languages*, an example of which is ALPHA.[22] This level of language is even less procedural than relational algebra. A very important characteristic of both algebraic and calculus languages is that any operation results in a new table. This means that composite expressions can be constructed in which the result of one operation become the operand of another.

display-oriented languages

A third class of relational languages is that of *display-oriented languages*, as exemplified by Query-By-Example (QBE). Here, instead of the relational operations being specified in a linear statement form directly as joins, selections, and projections, they are achieved by the manipulation of graphic symbols on a display screen.

network and hierarchical languages

There are also a few examples of query languages based on network or hierarchical data models.[18] However, experience shows that these are most effective when the predefined access paths in the data structure are used directly. When indirect access paths must be used, the query logic that must be specified by the user immediately becomes more complex even though the query itself appears to be simple. Therefore, predefined access paths sometimes appear as asymmetry and complexity to the user.

Notes on relational theory

When relational data base systems are studied theoretically, different terms are often found in the literature, as compared to busi-

ness data processing environments. This terminology makes the subject appear unnecessarily complex, and has contributed more than anything else to a misunderstanding of the concepts of tables.

The following is a list of terms with the formal name usually found in technical literature on the left and its everyday data processing equivalent on the right of the equivalency sign.

- Relation = table or record type.
- Tuple = row or record occurrence.
- Attribute = column name or field type.
- Element = field.
- Degree = number of columns in a table.
- Cardinality = number of rows in a table.
- Binary relations = table with two columns.
- N-ary relations = table with N columns.
- N-tuple = a record from a table with N columns.

There is no corresponding term for *domain*, but it has the following meaning. All values that may occur for a specific field type come from a domain of all the possible values of this type. Many different field types may use the same domain.

Even today, many authors of research articles use their own (and sometimes variant) definitions of relational terminology as a starting point for developing further ideas. This is a clear indication that there is still considerable evolution going on and that relational theory may mean different things to different people. We therefore conclude this section with the following more formal but well-established definition of a relation.

Given sets $S_1, S_2, \cdots, S_n$ (not necessarily distinct), R is a relation on these n sets if it is a set of n-tuples, each of which has its first element from S_1, its second element from S_2, and so on. More concisely, R is a subset of the Cartesian product $S_1 \times S_2 \times \cdots \times S_n$. S_j is the jth domain of R. R is said to have degree n.[1,23]

Relational design concepts

As with many evolving concepts, the idea of relational data bases breaks down into several areas, some of which are quite different and independent of one another. Preceding sections have discussed the tabular view of data, data access using such views, and languages that may be used. On this level, the relational approach is an alternative to a hierarchic[24] or a network approach.[25,26] Operational implementations of relational systems may be thought of as potential choices among DL/I[27] and CODASYL implementations.[28-30]

A quite different area of relational data base concepts deals with data base design theories, which cover the design of records. The theories are concerned with normalization and functional dependencies in record structures,[31] and are often presented with much mathematical formalism.

The prime objective of these theories is to help define data record structures that remain stable as the data base grows. Well-defined record structures avoid unnecessary future update problems and serve as a basis for future extensions. Existing record structures should not have to be restructured because of new application needs, although they may have to be extended, and new record structures may have to be added. But existing structures should survive such evolution without need for rearrangement of fields in existing record structures.

In this sense, design theories should apply to a number of data base management systems; a systematic design procedure is desirable, regardless of whether the resulting records are grouped into tables, hierarchies, or networks.[32] Normalization theory, for example, is not an issue in the realization of a set of record structures in certain data base management systems. Rather, the potential controversy lies in which data model is most suitable for the anticipated data access and manipulation of the record structures. Thorough data base design is thus a valuable and desirable practice for all three data models. The penalty for bad design is loss of data independence, the implications of which are clear to experienced users of data base management systems.

costs and benefits

With these basic concepts as a background, one might ask what is so dramatically new and useful in the relational view of data. An important potential drawback should be clear: performance for table operations may not be acceptable. It is not that relational data base systems are inherently less efficient in handling data requests than hierarchical and network implementations. On the contrary, they can make use of improved techniques in indexing and access methods. The problem is that performance may be experienced as being poor if the user is encouraged to do work of comparable complexity to that usually done with other data base systems without the same awareness of the required I/O operations, etc.

The price paid by a relational external interface is that there are no predefined access paths that the user can explicitly take advantage of, as in the hierarchical and network approaches. This does not mean that there could not be optimized paths under the cover in a relational data base implementation. Existing relational systems put much emphasis on providing access path optimization internally that is not made visible to the user.

It is also possible that developments in specialized hardware such as associative processors or logic-per-track devices might be especially suitable for efficient relational data access,[33] and might further improve the performance of relational data base operations. This, however, is not a practical or economical alternative.

We first consider potential benefits of the relational approach; then we consider the effect of that approach on different types of users. We conclude by identifying situations where a network or hierarchical data view is preferable to the relational view at the present time.

There are five major areas where the relational view has its strong points.

ease of understanding

Most persons have a common and intuitive idea of what a table is; the basic concepts are easy to understand. The concepts "common" and "intuitive" mean that the idea of a data base can potentially be more easily available to many more users than those who understand a CODASYL set or a DL/I logical data base hierarchy.

To some extent, complexity in such data base implementations as CODASYL or DL/I is caused by the multiplicity of different concepts and implementation constructs. This, in turn, depends on the asymmetrical ways of data access. Separate concepts are needed when predefined paths of different types are used. Examples are access to a hierarchy via a secondary index or access through a hierarchical path.

Up to the present time, there have been a limited number of relational data base implementations. One might imagine that in future implementations the simplicity of the high-level relational approach to data base access may be compromised by implementation particularities. A large, shared data base with many complex relationships among data items may need specialized facilities for certain crucial operations. However, the relational concepts are by their nature very straightforward and uncompromising in this respect. To a large extent, the simplicity we have seen so far in existing systems is an important part of the relational discipline itself.

increased data independence

The relational data base view deals directly and exclusively with rows and columns. All fields are explicitly known and seen by the user. Operations on tables do not depend on any predefined access paths that are implied in the data structure.

Neither do relational data structures depend on physical attributes of storage structures or on special implementation constructs like secondary processing sequence, concatenated logical

parent sequence fields, or incomplete path call conditions because there is a distinct boundary between the external data base view and the internal storage of data.

Our implied comparison with other implementations may not be realistic in the sense that it compares concepts of one with implementations of another. And practical implementations that must serve many different applications with a large, shared data base may require specialized language constructs to be efficient. Nevertheless, the relational approach provides a new chance to achieve a cleaner high-level interface.

power and ease of use

A major reason why relational operations are powerful and easy to use is that they operate as set access in contrast to record-at-a-time access. This means that relational operations become less procedural. Relational operators express more directly what the end result should be rather than describing how this end result should be produced. That leaves the data base management system instead of the user to perform retrieval and update operations at the detailed level. Less procedurality is a big step forward to increased productivity and high-level data base programming.

This characteristic becomes even more important in query applications where a user cannot be expected to specify in great detail how a particular question should be answered. Therefore, we foresee query applications as the first production environments for relational data base systems.

There is a parallel here involving comparisons of high-level programming languages such as COBOL and FORTRAN with more machine-oriented languages like Assembly and Autocoder. Today there is a fair agreement on the benefits of high-level programming languages, and there well may be the same type of agreement on the set-wise approach to data base access as compared to detailed record-at-a-time techniques.

theoretical foundation

Research work on relational operations has now gone on since the late sixties, and this subject is continuing to evolve.[34] A barrier to early application, however, is that much of this work is presented with a lot of mathematical formalism.[35] Practitioners are often sceptical of excessive formalism and mathematical notation. The theoretical foundation of relational systems, however, should not deter the practical and pragmatic data processing professional. It means that the results of relational operations are easily predictable; for instance, relational operations always produce the answer in the form of a new table.

In this regard, there is a clear distinction between relational systems and the more pragmatic data base management systems currently in use. Today's systems are the result of functions gradu-

ally extended or improved over a long time as the demand for additional or modified functions has increased. Therefore, some functions in present implementations are more ad hoc in nature and do not always fit nicely with previous concepts. The relational view of data, therefore, provides an opportunity for cleaner implementations of high-level data access.

Both users and implementors of data base management systems may benefit from this more theoretical basis for data base operations. For implementors, it means that a relational request may be more easily broken up into its component parts and rearranged, resequenced, and optimized. Intelligence may thus be transferred from individual program procedures to the data base management system.

With regard to data access, most emphasis has been placed on applications of retrieval theory and far less on the more complex operations of updating a data base. Increasingly, however, updating operations through relational views is receiving the necessary theoretical attention.[36,37] Thus, because of their potential usefulness, theoretical studies should be appreciated and encouraged.

table operations and data definition

The language of table operations used for data access may also be extended and generalized to data definition,[2,20] thereby allowing for common interaction among data base administrators, query users, and programmers. In contrast, different languages for data definition and for data manipulation are used in CODASYL and DL/I. This contributes to complexity and difficulties in communication among various user groups.

Even more important is the fact that the increased power and flexibility in data definition may also eliminate the need for some programming because the relational user view already expresses the data that are of particular interest to an application. Thus a more powerful data definition is substituted for programming, and the distinction between programming and data definition diminishes.

The programming effort may be further reduced because a relational data definition can allow one user's views to be expressed in terms of other users' views. Thus many levels of views-on-views are possible; a user view does not have to refer back to the stored data directly. This means that sometimes the underlying stored data may even change in structure without affecting many existing user views. Instead, a previously stored structure is replaced by a mapping of a new user's view. That new user view, in turn, refers back to the new stored structure. The concept of many levels of views-on-views is particularly powerful and valuable in achieving increased data independence. Also, the data

base system becomes more forgiving in that a previous data base design can be more easily modified with new user views. The restructuring of a previous data base design is simplified because new structures can be expressed in terms of older ones and added to the system gradually. Many slightly different user views can all be present at the same time, thereby reducing maintenance requirements.

Concluding remarks

It is often speculated that relational implementations will gradually replace network and hierarchic implementations. Such speculations seem too far-reaching. At the present time, certain applications seem to lend themselves most efficiently to a solution that uses tables as a data structure, whereas many others are best served by data hierarchies or networks. The potential value of relational data bases will probably not be the same for all types of users. For some, the benefit will be only marginal, whereas for others it may be significant.

end users

Users who are not data processing professionals—often termed end users—may see the greatest value in the tabular view. Such persons typically make unplanned query requests from data bases. Users sometimes find that present implementations are not completely successful in providing clear, precise, and simple language functions using hierarchies and networks. Similar queries must often be specified differently, depending on which predefined path in a hierarchy or network is used. The query language is thus asymmetrical. The relational view is designed to provide a symmetrical, simple, high-level interface for the query specifier. At the present time, however, specialized skill is needed to properly specify queries.

programmers

For conventional programmers, the value of relational systems strongly depends on the application. Often the same or a greater amount of programming logic is required for using a relational view than when the data base is seen as a hierarchy or network. This is particularly apparent when data have to be accessed one record at a time. An example is a bill-of-material application, where one has to follow the explosion/implosion loops individually. In other applications, a relational system may require many operations on multiple tables, whereas a hierarchical system may produce the desired result with a few operations that make more use of the implicit hierarchical structure of the underlying data.

In other applications, a relational view may simplify the programming logic, especially when set-oriented retrieval or updating is applied. In those cases, the simplification in logic makes the programs easier to understand and maintain and thereby contributes

to reduced maintenance costs as well. In set operations, the same operation may be applied to a number of rows in a table, as, for example, increasing all prices by a given percentage in a price table or changing all old locations to a new location in an employee table. Common among such operations is their property of being fairly simple and straightforward. Exceptions on an individual basis cannot be handled, but require instead record-by-record processing.

In some applications, however, gains made in a relational system by reduced procedurality may be lost in other ways by the requirement to work with many variations of tables and with a multitude of implied relationships among them. Because access paths are predefined and explicitly shown, a hierarchical or network diagram may capture in one quick glance an immediate understanding of many complex interrelationships. In contrast, it may often take greater time and effort to digest the same information using a large set of interrelated tables.

The programming of many applications should benefit from relational data definition capabilities, because data needed for a particular application can be more directly and precisely expressed to the program. This should eliminate requirements for the program to deal with those parts of a data base that are not of direct interest to the particular application. This apparent ease of use may also, to some degree, depend on such things as earlier programming background, education, and programming style.

data base administrators

For data base administrators the main problems are similar, regardless of whether they use a relational data base system. The administrator must still choose and define various storage options, maintain operation procedures, and monitor performance. Backup and recovery procedures must be maintained, and storage utilization must be monitored. It is possible, however, that the number of options and alternatives can be reduced in a relational implementation. One reason is the simplicity of the interface between external and internal definitions. Therefore, the data base management system may take over internally more and more of the functions handled by the data base administrator. That may imply, however, that useful implementation alternatives have been sacrificed for the sake of simplicity.

data base designers

The effort of doing data base design is expected to depend highly on the comprehensiveness of the data base management system. The more a system can take over the maintaining, reorganizing, and optimizing of access paths to the stored data, the less effort is necessary for a thorough data base design. A relational system has great potential in this area because of the clean and simple interface to the user.

Flexibility in data definition is expected to simplify the design effort considerably to accommodate new or changed data requirements, especially in small, private data bases. In private data bases, the data are often isolated from application to application, and performance implications are less important. To some degree, flexibility in relational data definition may also simplify designs of larger data bases that are then shared among many different applications and users.

When designing individual record structures, designers should not experience much difference among types of data base. Good design practices, such as normalization and elimination of dependencies among field types, are desirable regardless of whether the resulting record structures are used in a network, a hierarchical, or a relational system.

summary Relational data base systems present the user with simple, high-level data base processing. These systems incorporate features that complement network and hierarchical systems. The following are the five main complementary features of the relational approach:

- Table data structures are easy to understand.
- Tables provide increased data independence as compared to present implementations of hierarchies and networks.
- Table operations are powerful and still easy to use.
- Table operations have a sound theoretical foundation.
- Table operations may be generalized to data definition.

ACKNOWLEDGMENTS

Many thanks to Bill Kent and Dave Schofield for much constructive criticism of earlier versions of this paper.

CITED REFERENCES AND NOTES

1. E. F. Codd, "A relational model for large shared data banks," *Communications of the ACM* **13,** No. 6, 377–387 (June 1970). (See also Reference 23 for a listing of other early papers by this author.)
2. M. M. Zloof, "Query-By-Example: a data base language," *IBM Systems Journal* **16,** No. 4, 324–343 (1977).
3. *Query-By-Example Terminal Users Guide*, SH20-2078, IBM Corporation; available through IBM branch offices.
4. *Interactive Management and Planning System: User Guide*, SB11-5220; IBM Corporation, available through IBM branch offices.
5. M. M. Astrahan, M. W. Blasgen, D. D. Chamberlin, K. P. Eswaran, J. N. Gray, P. P. Griffiths, W. F. King, R. A. Lorie, P. R. McJones, J. W. Mehl, G. R. Putzolu, I. L. Traiger, B. W. Wade, and V. Watson, "System R: Relational approach to database management," *ACM Transactions on Database Systems* **1,** No. 2, 97–137 (June 1976).
6. S. Todd, "The Peterlee Relational Test Vehicle—a system overview," *IBM Systems Journal* **15,** No. 4, 285–308 (1976).
7. S. Todd, *Relational Database Research at the IBM UK Scientific Centre, Peterlee, a Survey 1970–1977*, Report UKSC-93 (December 1977); available from IBM United Kingdom Limited, Scientific Centre, Athelstan House, St. Clement Street, Winchester, Hants SO23 DR9, England.

8. W. Kim, "Relational database systems," *ACM Computing Surveys* **11,** No. 3, 185-211 (September 1979).
9. R. G. Ross, "Assessment of current data base trends," *Data Base Monograph*, No. 5, R. M. Curtice, Editor, Q.E.D. Information Sciences, Wellesley, MA (1977).
10. This is the only normalization requirement for a relational data base. The 2nd, 3rd, and 4th normal form all represent improved qualities in record structures. These normal forms, however, are not required for implementation in a relational data base.
11. A join may also be performed on two columns in the same table as well as on multiple columns simultaneously.
12. S. B. Yao, "Optimization of query evaluation algorithms," *ACM Transactions on Database Systems* **4,** No. 2, 133-155 (June 1979).
13. M. W. Blasgen and K. P. Eswaran, "Storage and access in relational data bases," *IBM Systems Journal* **16,** No. 4, 363-377 (1977).
14. T. Haerder, "Implementing a generalized access path structure for a relational database system," *ACM Transactions on Database Systems* **3,** No. 3, 285-298 (September 1978).
15. J. P. Fry and E. H. Sibley, "Evolution of data-base management systems," *ACM Computing Surveys* **8,** No. 1, 7-42 (March 1976).
16. C. J. Date, "An architecture for high-level language database extensions," *Proceedings of the 1977 SEAS Anniversary Meeting*, Cambridge, England, 315-420 (1977).
17. In the case of DL/I path calls, the result of a retrieval may be a few records along the hierarchical path instead of just one single record.
18. H. Lehman and A. Blaser, *Query Languages in Data Base Systems*, Research Report TR79.07.004, IBM Heidelberg Scientific Center, Tiergartenstrasse 15, 6900 Heidelberg, Germany (1979).
19. A. Pirotte, "Fundamental and secondary issues in the design of non-procedural relational language," *Proceedings of the 5th International Conference on Very Large Data Bases*, Rio de Janeiro, October 1979, 239-250 (1979); available from the Association for Computing Machinery, 1133 Avenue of the Americas, New York, NY 10036.
20. D. D. Chamberlin, M. M. Astrahan, K. P. Eswaran, P. P. Griffiths, R. A. Lorie, J. W. Mehl, P. Reisner, and B. W. Wade, "SEQUEL 2: A unified approach to data definition, manipulation, and control," *IBM Journal of Research and Development* **20,** No. 6, 560-575 (November 1976).
21. M. W. Blasgen, M. M. Astrahan, D. D. Chamberlin, J. N. Gray, W. F. King, B. G. Lindsay, R. A. Lorie, J. W. Mehl, T. G. Price, G. R. Putzolu, M. Schkolnick, P. G. Selinger, D. R. Slutz, H. R. Strong, I. L. Traiger, B. W. Wade, and R. A. Yost, "System R: An architectural overview," *IBM Systems Journal* **20,** No. 1, 41-62 (1981, this issue).
22. E. F. Codd, "A data base sublanguage founded on the relational calculus," *ACM SIGFIDET Workshop on Data Description, Access, and Control*, San Diego, CA, 35-68 (1971).
23. D. D. Chamberlin, "Relational data base management systems," *ACM Computing Surveys* **8,** No. 1, 43-66 (March 1976).
24. D. C. Tsichritzis and F. H. Lochovsky, "Hierarchical data base management: A survey," *ACM Computing Surveys* **8,** No. 1, 105-123 (March 1976).
25. T. W. Olle, *The Codasyl Approach to Data Base Management*, John Wiley & Sons, Inc., New York (1978).
26. R. W. Taylor and R. L. Frank, "CODASYL data base management systems," *ACM Computing Surveys* **8,** No. 1, 67-103 (March 1976).
27. W. C. McGee, "The information management system IMS/VS; Part II: Data base facilities," *IBM Systems Journal* **16,** No. 2, 96-122 (1977).
28. *Series 600/6000 Integrated Data Storage Reference Manual*, CPB-1565; Honeywell Information Systems, Incorporated, Honeywell Plaza, Minneapolis, MN 55408.
29. *UNIVAC 1100 Series, Data Management System (DMS 1100) Schema Definition*, UP-7907; Sperry Univac, P.O. Box 500, Blue Bell, PA 19422.

30. *Integrated Database Management System Program and Reference*, Cullinaue Corporation, 20 William St., Wellesley, MA 02181.
31. C. Beeri, P. A. Bernstein, and N. A. Goodman, "A sophisticated introduction to database normalization theory," *Proceedings of the 4th International Conference on Very Large Data Bases, West Berlin, September 1978*, 113–124 (1978); available from the Association for Computing Machinery, 1133 Avenue of the Americas, New York, NY 10036.
32. J. J. Janko, "Relational design of an IMS database," *Database*, Online Conferences, Limited, Uxbridge, Middlesex, England (1977).
33. G. G. Langdon, "A note on associative processors for data management," *ACM Transactions on Database Systems* **3,** No. 2, 148–158 (June 1978).
34. E. F. Codd, "Extending the database relational model to capture more meaning," *ACM Transactions on Database Systems* **4,** No. 4, 397–434 (December 1979).
35. A. V. Aho, C. Beeri, and J. D. Ullman, "The theory of joins in relational databases," *ACM Transactions on Database Systems* **4,** No. 3, 279–314 (September 1979).
36. F. Bancilhon, "Supporting view updates in relational data bases," *Proceedings of the IFIP Working Conference on Data Base Architecture*, Venice, Italy, June 1979, North-Holland Publishing Company, Amsterdam (1979), pp. 213–234.
37. U. Dayal and P. A. Bernstein, "On the updatability of relational views," *Proceedings of the 4th International Conference on Very Large Data Bases*, West Berlin, September 1978, 368–377 (1978); available from the Association for Computing Machinery, 1133 Avenue of the Americas, New York, NY 10036.

GENERAL REFERENCES

C. J. Date, *An Introduction to Database Systems*, Second Edition, Addison-Wesley Publishing Company, Reading, MA (1977).

1978 New Orleans Database Design Workshop Report, Proceedings of the 5th International Conference on Very Large Data Bases, Rio de Janeiro, October 1979, 328–339 (1979); available from the Association for Computing Machinery, 1133 Avenue of the Americas, New York, NY 10036.

W. Kent, *Data and Reality, Basic Assumptions in Data Processing Reconsidered*, North-Holland Publishing Company, Amsterdam (1978).

M. E. Senko, "Data structures and data accessing in data base systems past, present, future," *IBM Systems Journal* **16,** No. 3, 208–257 (1977).

J. P. Fry and E. H. Sibley, "Evolution of data base management systems," *ACM Computing Surveys* **8,** No. 1, 7–42 (March 1976).

A. S. Michaels, B. Mittman, and C. R. Carlson, "A comparison of the relational and CODASYL approaches to data-base management," *ACM Computing Surveys* **8,** No. 1, 125–150 (March 1976).

C. J. Date and E. F. Codd, "The relational and network approaches: Comparison of the application programming interface," *ACM SIGMOD Workshop on Data Description, Access, and Control*, Ann Arbor, MI, R. Rustin, Editor, 11–41 (1974).

D. Smith and J. Smith, "Relational database machines," *Computer* **12,** No. 3, 28–38 (March 1979).

S. Su, "Cellular-logic devices: Concepts and applications," *Computer* **12,** No. 3, 11–25 (March 1979).

The author is located at the IBM Nordic Field Systems Center, Oddegatam 5, Kista, S-163 92 Stockholm, Sweden.

Reprint Order No. G321-5139.