DATA MODELS in DATABASE MANAGEMENT

E. F. Codd
IBM Research Laboratory
San Jose, California 95193

1 WHAT IS A DATA MODEL?

It is a combination of three components:

1) a collection of data structure types (the building blocks of any database that conforms to the model);
2) a collection of operators or inferencing rules, which can be applied to any valid instances of the data types listed in (1), to retrieve or derive data from any parts of those structures in any combinations desired;
3) a collection of general integrity rules, which implicitly or explicitly define the set of consistent database states or changes of state or both -- these rules may sometimes be expressed as insert-update-delete rules.

Note that in any particular application of a data model it may be necessary to impose further (application-specific) integrity constraints, and thereby define a smaller set of consistent database states or changes of state. Note also that a database system must normally permit states other than the consistent ones to exist transiently during the execution of a program. It is imperative that the program tell the system at which steps it is permissible for the system to check integrity. There may exist programming languages which permit the intermixing of integrity assertions and commands, but I do not know of any (other than database sublanguages) which permit the specification of integrity points at which a set of community-specified integrity rules are to be checked.

Numerous authors appear to think of a data model as nothing more than a collection of data structure types. This is like trying to understand the way the human body functions by studying anatomy but omitting physiology. The operators and integrity rules (items 2,3 in the definition above) are essential to any understanding of how the structures behave. In comparing data models people often ignore the operators and integrity rules altogether. When this occurs, the resulting comparisons run the risk of being meaningless.

A flagrant example of such a comparison is the statement in a panel discussion on Standards in ACM SIGMOD 1979 (recorded in the Supplement to the Proceedings, page 55): "the relational model is considered to be a constrained version of the flat file data model." What are the operations that are allowed on flat files? What are the general integrity constraints on flat files? Is there even a generally accepted definition of the structure of flat files that is sufficiently precise so that we can tell for sure whether a flat file can contain records of more than one type?

Note that the authors of many of the data models of the past five years defined the data structures only, omitting the operators and integrity rules. Such models should therefore be regarded as partial or incomplete data models.

2 PURPOSES OF A DATA MODEL

A data model may be used in any of the following ways:

1) as a tool for specifying the kinds of data and data organization that are permissible in a specific database;
2) as a basis for developing a general design methodology for databases;
3) as a basis for coping with evolution of databases so as to have minimal logical impact on existing application programs and terminal activities;
4) as a basis for the development of families of very high level languages for query and data manipulation;
5) as a focus for DBMS architecture;
6) as a vehicle for research into the behavioral properties of alternative organizations of data.

Re item 4), a data model need not (and probably should not) dictate a single language for data manipulation and query, since different kinds of users are likely to need different kinds of languages. The operators or inference rules should, however, provide a yardstick of manipulative and query power.

The extent to which data models have influenced the field of database management can be seen by observing the new database systems (experimental and product) that have been developed during the last ten years. It is hard to find one that is not based on either the CODASYL network model or the relational model. The number of CODASYL implementations and installations is often attributed (and I think correctly) to a desire to conform to a committee-defined data definition language. However, this raison d'etre certainly does not apply to existing relational systems.

The increasingly widespread use of the relational model as a vehicle for logical database design (regardless of the target database management system by which the data is to be ultimately managed) provides additional evidence of the impact of data models on the database field. Substantial developments in the theory of database structure have been triggered by the work on normalization of relations in the relational model.

The relational model has also spurred vigorous and widespread research into techniques for optimizing the execution of statements in very high level database languages. Other models are seldom, if ever, used for such investigations, because their high level languages (when such exist -- and I know of only one that has been implemented) are necessarily more complicated.

Finally, it appears that database models have influenced programming language research, providing early examples of data abstractions. Data models have paved the way for the much clearer separation of semantic issues from implementation issues in programming languages. Data models can also be expected to bring about a belated recognition that general purpose programming languages need to distinguish shared variables from private variables.

3 HISTORY OF DATA MODEL DEVELOPMENT

As of 1979, some 40 or more data models (mostly incomplete in the sense defined above) have been proposed for the management of formatted data. The first such data model to be developed was the relational model (developed in 1969). Many people have the erroneous impression that the hierarchical and network models preceded the relational model. This is due to a confusion between language specification and implementation on the one hand and data models on the other. Hierarchical and network systems were developed prior to 1970, but it was not until 1973 that data models for these systems were defined. It is a little known fact that the hierarchic model (incomplete as it is) was defined by a process of abstraction from IBM's IMS. Similarly, the network model (incomplete as it is) was defined by abstraction from the CODASYL DBTG language proposals of 1969. The purpose of these definitions was to provide a basis for comparing the three approaches on a common level of abstraction. Thus, hierarchic and network systems preceded the hierarchic and network models, whereas relational systems came after the relational model and used the model as a foundation.

4 COMMON MISUNDERSTANDINGS

Many people fail to separate in their minds different levels of abstraction. A specific example of this is the failure to realize that tuples are at a higher level of abstraction than records (one is not allowed to use the contiguity of components of tuples, whereas one can use the contiguity of fields in a record).

Likewise, primary keys (whether they have system-controlled surrogates or user-controlled identifiers as values) are at a higher level than pointers. A particular occurrence of a value V of a primary key makes reference to all other occurrences of V in the database that are drawn from the domain of that primary key. Surrogates have the property that they are distinct if they represent distinct objects in the real world. They are at a higher level than DBTG database keys, which are record identifiers that are distinct for distinct records. Note that there may be two or more records describing a single real world object, in which case there are two or more database keys corresponding to one surrogate. Moreover, within one record there may be two or more surrogates and only one database key.

Another kind of confusion concerns the exclusion of relations of degree higher than two from the principal schema. This exclusion is sometimes claimed to remove all concern for anomalies in insertion, update, and deletion. To see that this claim is false one need only compare two alternative anchored binary schemas for an n-ary relation R that is known to possess such anomalies. In the first schema there are n binary relations, each corresponding to one of the n attributes (columns) of R. In the second schema R is first of all non-loss decomposed (by projection, for example) into two or more relations of lesser degree, and then these relations are converted to anchored binary form. These two schemas give rise to databases with entirely different insertion, update, and deletion behavior.

Whether binary relations (carefully defined with due regard to possible anomalies) are better than relations of higher degree (similarly carefully defined) is a separate question which can be argued at length. My present position is that this is largely a subjective question. The differences between these two views of data are not significant for formatted databases, in which the data exhibit a great deal of regularity. Moreover, operators that generate and manipulate n-ary relations are unavoidable if one is to support a variety of user views and a variety of queries.

A common error is to confuse the concepts of attribute (column) and domain (the set of all those values which can ever occur in a given column). This is perhaps due to the fact that there is no counterpart to the domain concept in the most widely used programming languages. PASCAL may be the first programming language to incorporate some aspects of the database domain concept. In a highly shared and dynamic environment it is important that the system keep track of which columns of which tables draw their values from any given domain. Otherwise, it is impossible to write a reliable program to remove from a database all references to a particular entity (see Appendix).

Another error is that of identifying the join operator of the relational model with the links (sometimes known as fan sets) of the DBTG model. These concepts are different in ways too numerous to mention. Suffice it to say that, while the relational join operates on a pair of tables to yield a table as the result, DBTG links are not operators, but merely structures that by themselves yield nothing (operators must act upon them if any information is to be extracted from them or per them).

Different users need to see the database in different ways. As Smith and Smith point out, the concepts of entity, relationship, property, category, and even database value represent different perceptions of common abstract objects. A data model that does not permit a relationship to be viewed as an entity is clearly inadequate to support these different perceptions (several such models have been proposed).

Recent investigations into semantic data models represent an important contribution to the understanding of the meaning of data in formatted databases. However, this work is sorely in need of some objective criteria for completeness: i.e., knowing when to stop. At present, this is a matter of taste.

5 THE FUTURE

The subject of data modeling will be a fertile area for research, development, and application for many years to come. This is due principally to the fact that the meaning of data and the manipulation of this meaning are still so poorly understood. Further, the impact of data modeling on database management will continue to be high, affecting both the design of databases and the design of database management systems. Gradually, designers are becoming aware of the need for very high level data sublanguages:

1) to support efficient communication of data between distributed databases;
2) to enable the system to determine access strategy in the face of data representation which is subject to dynamic change from time to time (System R does this).

If a user at one node needs a collection of records from another node and he is able to specify that collection in a single statement, it is absurd for him to engage in a sequence of single record requests, each followed by single record replies. One reason the relational model is in such a dominant position today is that, when originally introduced, two radically different, very high level (set-oriented) data sublanguages (the relational algebra and predicate-logic-based ALPHA) were defined for it. There are now approximately twenty such data sublanguages for the relational model. By the end of the 80's it is reasonable to expect the relational model to have outstripped every other data model in terms of the number of users, the number of databases, and the number of database systems.

APPENDIX: The crucial role of domains

An important part of keeping a relational database in a state of integrity is keeping track of which attributes (columns) are defined on which domains. This information is needed to support the global removal of all occurrences of a value V where it occurs as a value from domain D. Here, we are referring to domain in a semantic, not a syntactic, sense. That is, we wish to discuss domains such as supplier serial numbers, quantities of parts, names of projects, rather than domains such as alphanumeric character strings, floating point numbers, and integers.

For example, we may wish to remove supplier Jones from the database. He happens to have the serial number 3, and we want to remove his serial number and descriptive properties from the SUPPLIER relation, but in addition we want to remove all occurrences of 3 as a supplier serial number in all other relations. These latter occurrences are called referential occurrences, and are to be replaced by null, except where the integrity constraints demand that a null is unacceptable, in which case the tuple containing the component that references supplier 3 is to be deleted.

Suppose that on a certain day a programmer writes a program to carry out the removal of any specified supplier, and this program is based on his knowledge as of that time concerning which columns are defined on the supplier serial number domain. Such a program will fail to operate correctly, if at a later time one or more new tables are created that have columns defined on this domain. Clearly, for such a program to operate correctly regardless of changes that may be made in the tables referencing the supplier serial number domain, the database system must have the knowledge regarding which columns use which domains -- and this knowledge must be kept up-to-date by the system every time a new table is created, extended (by adding a new column), or destroyed.

When a new table is declared, accompanying each column name should be the name of the semantic domain on which that column is defined. The system should keep this information in the column-domain table of the database catalog. The addition of a new column should be similarly treated. Corresponding deletions in the column-domain table should occur as a side effect of any command that drops tables.

The proposal that all columns on a given domain be identically named throughout the database is not a feasible solution, since any table may have more than one column defined on the given domain.