Limitations of Record-Based Information Models

WILLIAM KENT
IBM Corporation

Record structures are generally efficient, familiar, and easy to use for most current data processing applications. But they are not complete in their ability to represent information, nor are they fully self-describing.

Key Words and Phrases: records, information model, data model, semantic model, conceptual model, normalization, first normal form, entities, relationships
CR Categories: 3.70, 4.33, 4.34

INTRODUCTION

Records provide an excellent tool for processing information that fits a certain pattern. Other kinds of information do not fit as well into record structures. In all cases, the use of record structures depends on supplementary information, often reflected only in the special-purpose application programs written to process the data, and which may or may not still be remembered by the users of the data. Record structures do not provide the semantically self-describing base needed for conceptual schemas [2, 3, 21], or for generalized query processors or other end-user facilities.

In their capacity as data processing *tools,* records have a desirable versatility. That is, a given construct (e.g., field names, or compound fields) can be used for different purposes at different times. Unfortunately this virtue becomes a vice for semantic modeling: one has to know the special usage of each construct in each case, and there is no general rule for deducing the underlying semantic structure. While some information cannot be represented in records, other information can be represented in so many ways as to become ambiguous.

Models which provide additional file structure around the records (e.g., sequencing, hierarchies, CODASYL networks) overcome some of the functional limitations. None of them overcome all the limitations. Furthermore, by building on top of record structures, they retain all the underlying ambiguities. In some cases, they simply add more options for representing something which could already be represented in several ways in record structure.

This paper would be pointless if there were not any alternatives, but there are.

Author's address: General Products Division, IBM Corporation, 555 Bailey Avenue, P.O. Box 50020, San Jose, CA 95150.

There are various models which are essentially graph structured, based on such primitive concepts as binary relations, or entities and relationships. Such models tend to be more functionally complete in their information processing capability, and more precise in their semantic modeling. (We are not arguing their economic superiority to record structures for bulk data processing in today's applications.) A number of these are listed in the concluding section of this paper. We do not try to explain or defend these models. Our main purpose is to collect some problems concerning record structures (many of which have already been mentioned in the literature), as a resource to help defend alternative models. Nevertheless, much of the discussion is cast in terms of entities and relationships. The motivation for this paper originated in attempts to reconcile record structures with the characteristics of entities and relationships.

"RECORD" DEFINED

By *record* we mean here a fixed sequence of field values, conforming to a static description usually contained in catalogs and/or in programs. The description consists mainly of a name, length, and data type for each field. Each such description defines one *record type* (or, in relational terms, a "relation" or "table"). My remarks apply to any data model based on this kind of construct. This clearly includes the traditional hierarchical, relational, and (CODASYL) network models. (For introductions to these models, see [7, 9-13, 42, 43].) It also includes approaches mentioned in [2] and [18], to the extent that they speak of such things as "conceptual records." (It should be noted that such constructs are not mentioned in [3].) The comments also apply to the entity-relationship model of [8], which is really driven by record structures (relations) rather than by entities and relationships as the primitive concept (his Level 1 constructs appear to be constrained to match his Level 2 information structure).

Some record formats allow a certain variability by permitting a named field or group of fields to occur more than once within a record (i.e., as a list of values or sets of values). We will use the term *normalized system* to refer to systems which do not permit repeating groups or fields. This follows the relational model, which excludes such repetitions via its normalization requirements (specifically, first normal form [9, 20]).

BASIC ASSUMPTIONS BEHIND RECORD STRUCTURES

Record structure presumes a horizontal and vertical homogeneity in data: horizontally, each record of a given type contains the same fields; and vertically, a given field contains the same "kind" of information in each record.

Homogeneity of Relevant Facts

The records of a given type in a file describe a set of things in the real world (e.g., employees). Record structure fits best when the entire population has the same kinds of attributes (e.g., every employee has a name, address, department, salary, etc.). While exceptions are tolerated, the essential configuration is that of a homogeneous population of records, all having the same fields.

Although commercial data processing naturally focuses on areas which fit this pattern, the pattern does not always hold. In many cases, although a certain group of individuals constitutes a single "kind" of thing, there is considerable variation in the facts relevant to each individual in that set.

Consider clothing. While we can agree that pants, socks, underwear, and hats are all items of clothing, it would be very hard to define a conventional "clothing" record type. There are many field names which are relevant; not many of them apply to any one kind of clothing. Consider: size, waist size, neck size, sleeve length, long or short sleeves, cup size, inseam length, button or zipper, sex, fabric type, heel size, width, color, pattern, pieces, season, number, collar style, cuffs, neckline, sleeve style, weight, flared, belt, waterproof, formal or casual, age, pockets, sport, washable, etc. How would you design the record format for clothing records?

Clothing is by no means the only such category. Tools, furniture, vehicles, and people are just a few other categories having inhomogeneous attributes over their populations.

The more that information deviates from the norm of homogeneity, the less appropriate is the record configuration. There are certain techniques for accommodating variability among instances in a record structure, but these need to be used sparingly. If there is considerable variation over a population, then the solutions become cumbersome and inefficient. Such solutions include:

(1) Define the record format to include the union of all relevant fields, where not all the fields are expected to have values in every record. (Often "maiden name" is defined as part of a record format for all employees, though it is only relevant to married females.) Thus many records might have null values in many fields. Furthermore, the limited relevance is not defined to the system; it is only the pattern of usage (and, sometimes, validation logic in programs) which reflects the limitation.

(2) Allow the same field to have different meanings in different records. Unfortunately, such a practice is never defined to the system. With respect to any processing done by the system, that field appears to have the same significance in every record occurrence. It certainly has only one field name, which in these cases usually turns out to be something totally uninformative, like CODE or FIELD 1. It is only the buried logic in application programs which knows the significance of these fields, and the different meanings they have in different records.

Such inhomogeneity is especially vexing if it affects an attribute we would like to use as an identifier. If it does not apply to all individuals in the set, then it cannot be used as a "key" for the record type. This situation is fairly common. The employees of a multinational corporation might not all have social security numbers, or employee numbers. Some books do not have "International Standard Book Numbers" (ISBN), others do not have Library of Congress numbers, and some have neither. Library of Congress numbers are also given to things which are not books (e.g., films and recordings); those would not have ISBN's. Oil companies have their individual conventions for naming their own oil wells, and the American Petroleum Institute has also assigned "standard" names to some wells—but not all.

The function needed here is equivalent to "self-naming" fields, i.e., redefining the concept of record to mean a chain of relevant field names and their values.

Homogeneity Within Fact Type

That was a kind of "horizontal" homogeneity, i.e., each record containing the same fields. There is also a "vertical" homogeneity assumed. Within a record type, a given field is expected to contain the same kind of value in every record, as though a certain kind of fact always involved the same kinds of objects.

Again, this is not always true. Suppose that company cars can be assigned either to employees or to departments. If employees are normally identified by six numeric digits and departments by four alphanumeric characters, how do we design a "vehicle assignment" record? Assignment is a simple fact, to which one might naively expect to be able to address a simple inquiry: "to whom is car 97 assigned?" In such a case, we might like a two-part answer: the type of the assignee (employee or department), plus the identification of the individual assignee (in a format which depended on the assignee type).

We could design a record format with four fields: vehicle number, assignee type, employee, and department. The second field would tell us the assignee type, and hence whether to look for the assignee name in the third or fourth field. We assume, of course, that only one of the last two fields is filled in—but there is not likely to be any system facility to enforce that. Thus the multifield format introduces a data integrity hazard. And, as far as the record definitions convey any meaning to the system, we have here three independent facts about vehicles, with no interdependence among the three. The data structure bears little resemblance to the semantic structure of the underlying relationships. (We have, incidentally, created a horizontal inhomogeneity. The employee field is relevant for some vehicles, and the department field for others. They are never both relevant for the same vehicle.)

If, later on, cars can be assigned to other kinds of things with different identifier formats (e.g., to divisions, or to branch offices) then the record formats have to be redesigned with additional fields for the new assignee types. The validation gets more complicated: only one of the last *n* fields may contain a value. And the file may have to be physically reloaded for each format change.

Another approach is to provide distinct record types, one for each type of assignee. The fields in one record type would be vehicle number and employee, in another they would be vehicle number and department, and so on. Each record type has its own name; instead of "vehicle assignment" being a single kind of fact, we have many kinds: "vehicle-employee assignments," "vehicle-department assignments," etc. (One is required to believe that employees and departments cannot have the same relationships with vehicles. A relationship between an employee and a vehicle is necessarily "different" from a relationship between a department and a vehicle.) Instead of going to one record type (or naming one relationship) to find the assignment of a vehicle, one now has to know how many such record types there are—and their names—and be prepared to interrogate each one of them. It is even worse if you are interested in some other information about the vehicle, not its assignment. That information might be in any one of the record types. Validation is still a problem: there is no system facility to keep the same vehicle from appearing in more than one record type (i.e., having more than one assignee). Extensions are difficult, too: every new assignee type requires the introduction of another record type. And changing a vehicle's assignment is cumbersome, if the assignee type is also changing; a record of one type has to be deleted and a record of another type inserted.

Neither multifield records nor multiple record types offer a good solution to the vertical inhomogeneity problem. These approaches look even worse if there is inhomogeneity on both sides of the relationship. Suppose that instead of vehicle assignments, we were recording more general equipment assignments. Assignable equipment might include vehicles, furniture, tools, etc., each potentially having its own identifier formats. If there are m kinds of assignable equipment and n kinds of assignees, then the multifield approach requires $m + n + 2$ fields (two type fields), with only four fields containing values in any one record. The multiple record type approach would require $m \times n$ record types.

Still another solution to vertical inhomogeneity is to make it disappear. One way is to relax the field definition to a level that is general enough to handle all necessary identifier formats; a varying character string would do the job. Vehicle assignment records are then reduced to two fields again, where the second field might contain employee numbers, department codes, or the identifiers for any other kinds of assignees. Only users (and the code in their programs) know which is which, and what to do with them. The system is unable to furnish services such as syntax checking of the field values, or following a path to the corresponding employee or department record (by matching key values, as in the relational join), or verifying that a referenced employee or department does, in fact, exist.

This solution also fails if things in different categories might accidentally have the same identifiers (e.g., in the general equipment assignments, if a vehicle registration number might happen to be the same as a tool inventory number). Which points out that the vertical inhomogeneity problem is not simply a record format problem. Even if the record formats are compatible (e.g., employees and departments both having four-character codes), one has to guard against different entity types occurring in the same field if they might have the same identifiers.

But on the other hand, we do not have to have multiple entity types to encounter vertical inhomogeneity. Identifier formats can vary even within a single entity type. Employee numbers might differ in various subsidiaries of a corporation, or within a multinational corporation. Ship registry formats differ according to the country of registry. Oil companies have different formats for identifying their oil wells. The soldiers in a United Nations military group are likely to have different kinds of serial numbers. And so on.

There is still another way to force the disappearance of vertical inhomogeneity. One can provide a uniform reference to all the entities involved by aggregating them into one "supertype" and giving them a new arbitrary identifier, e.g., an "assignee number." (Analogous to such familiar constructs as "taxpayer identification number," or "capital equipment inventory number.") This permits assignees to be referenced uniquely and carefully in the assignment records, with a well-defined and checkable identifier format. Unfortunately, all of the entities involved have now acquired a new and additional identifier for which values have to be assigned, and by which they have to be recognized in various contexts. And, in some cases a readable name (perhaps the department name) is replaced by an unintelligible code number, which has to be looked up somewhere else.

What could such an identifier represent? In a record structure, it should be the key of a record type, i.e., an "assignee" record type. But if we also wanted to have employees and departments represented in distinct record types, we have a conflict. The type of a record is either "department" or "assignee"; it cannot be both.

Still another drawback of this "supertype" approach is that it has to be reapplied for each different kind of multitype fact (i.e., potentially for each case of vertical inhomogeneity). Entities get aggregated one way for equipment assignment records, another way to keep track of who manufactures what, another way for who owns what, another way as "employers" (which might be people, companies, schools, government agencies, foreign organizations, etc.), perhaps still another way as "taxpayers," and so on. Each of these is potential grounds for another supertype, with its own identifier scheme. This touches on the problem of multiple types for an individual, which we will get to later; the immediate concern is that an individual might become attached to a great many serial numbers, potentially one for each aggregation to which it belonged.

Vertical inhomogeneity can introduce still another record formatting problem. The identifiers involved may differ in more than just length and character set (e.g., numeric vs. alphabetic). There may be differences in "structure," e.g., if qualified naming is involved. Consider a company in which the name of a department is unique within its division, but not necessarily within the company as a whole. Then corporate records would have to refer to a department using two fields: a department name plus a division name (serving as a qualifier for the department). At this point, it is not at all clear how many fields there are in a corporate vehicle assignment record. If assigned to an employee, there are two: vehicle number and employee number. When the assignee is a department, there are three fields: vehicle number, department name, and division name. (If we had to describe this relationship in terms of the relational model, we might have to call it a relation of degree two and a half, on the average.)

To net it all out, the record structure is not well suited to information exhibiting "vertical inhomogeneity." The function required here is equivalent to allowing a single fact (field) to include both a type and value, where the syntax and structure of the value depended on the type.

PRESUMPTIONS UNDERLYING TRADITIONAL IMPLEMENTATIONS

Although not intrinsic to the record structure, a number of features characterize most traditional implementations of record processing systems. These include such things as the separation of descriptions and data, minimal requirements for descriptions, and resistance to changing descriptions.

Descriptions Are Not Information

Information is obtained from a record structure by extracting the values of fields, and it is only field values which supply information. One can answer the question "who manages the Accounting department?" by finding a certain field which contains the manager's name. But it is not likely that the file can provide an answer to "how is Henry Jones related to the Accounting department?" There are no fields in the file containing such entries as "is assigned to," "was assigned to," "on loan to," "manages," "audits," "handles personnel matters for," etc. Depending on how the records are organized, the answer generally consists of a field name or a record type name, which are not contained in the records. To a naive seeker of information from the database (e.g., via a high-level query interface), it is not at all obvious why one question may be asked and the other may not.

It is not just that he cannot get an answer; the interfaces do not provide any way to frame the question. The data management systems do not provide a way to ask such questions whose answers are field names or record type names.

Then consider the following questions:

(1) How many employees are there in the Accounting department?
(2) What is the average number of employees per department?
(3) What is the maximum number of employees currently in any department?
(4) What is the maximum number of employees permitted in any department?
(5) How many more employees can be hired into the Accounting department?

If the maximum number of employees permitted is fixed by corporate policy, then a system offering advanced validation capabilities is likely to place that number into a constraint in a database description, outside the database itself. Our naive seeker of facts will then again find himself unable to ask the last two questions. He might well observe that other things having the effect of rules or constraints are accessible from the database, such as sales quotas, departmental budgets, head counts, safety standards, etc. The only difference, which does not matter much to him, is that some such limits are intended to be enforced by the system, while others are not. It is not at all obvious to him why he can ask some questions and not others.

This suggests that we should represent such descriptions and constraints in the same format—and in the same database—as "ordinary" information, but with the added characteristic that they are intended to be executed and enforced by the data processing system.

There is, of course, an inherent difference between descriptions and other data, with respect to update characteristics. Changes to descriptions imply differences in the system's behavior, ranging from changes in validation procedures to physical file reorganizations implied by format changes. Thus the system has to be aware of, and control, changes to descriptions. But such descriptions need not be inherently different for retrieval purposes. And even with respect to update, the method need not be inherently different as perceived by users. It is only necessary that the authorization to do so be carefully controlled, and that the consequences be propagated into the system. There is already a precedent for such update controls: many implementations forbid the modification of key fields of records.

Some Descriptions Are Not Needed

While such things as field names and record types can be factored out of the data [32], they do not always wind up in the catalogs (record descriptions) either. Sometimes they do not appear anywhere in the data management system at all.

Catalogs are maintained primarily for the benefit of the system, not for users, and tend to contain only such information as is needed for the performance of system services. Quite often, only key fields are described, for which the system may provide such services as indexing, ordering, and uniqueness checking. Other fields might only be described in the declarations local to the various application programs, with no assurance that such descriptions are consistent with each other.

One of the major contributions of the relational model is to treat all fields in a record as constructs requiring description to the data management system.

Field Names Are Only Place Holders

When provided at all, field names are used by record management systems only to designate some space within a record. This suffices for the system to provide its services, such as matching keys or sequencing. And, certainly, one name is adequate for this purpose. But for information modeling, we may want to attach several labels to a field, indicating perhaps the kind of entity which may occur there (e.g., "date") and its relationship to the subject of the record— its reason for occurring in the record (e.g., "termination").

In practice, there has been no discipline in the usage of field names. Sometimes they name the entity type, sometimes the relationship, sometimes a hybrid of the two ("termination-date"), sometimes an identifier type ("social security number"), and sometimes nothing intelligible ("code 1," "field *x*"). And even when they do name entity types or relationships, field names are just mnemonic aids to human users, rather than anything which can be used by a system service to establish semantic connections. If the field name specifies the entity type, it is not likely to be the same as the corresponding record type name (while a record type might be named "dept-rec," the corresponding field in an employee record might be named "deptno"). The same entity type might be spelled differently in different field names ("dept," "deptnum," "deptno," etc.). And nothing prevents the same field name from meaning entirely different things in different records.

The relational model does improve on this situation by providing for both "selector" (column) names and "domain" names, but there is still relatively little discipline imposed. Domain names often specify identifier types rather than entity types [19]. Thus "social security number" and "employee number" are likely to be specified as the domains of two fields, giving no clue that the same entity might be named in both places. Even if the domain name identified an entity type, it might or might not be the same name as the record type representing those entities. And domain names give no clue when one entity type is a subset of another; unequal domains appear to be disjoint.

Some implementations of the relational model do not incorporate the domain construct at all.

Stability of Relevant Facts

Another implication of record formats, and of the file plus catalog configuration, is that the kinds of facts relevant to an entity are predefined and are expected to remain quite stable. It generally takes a major effort to add fields to records.

While this may be acceptable and desirable in many cases, there are situations where all sorts of unanticipated information needs to be recorded, and a more flexible data structure is needed.

The need to record information of unanticipated meaning or format is crudely reflected in provisions for "comments" fields or records. These consist of unformatted text, in which system facilities can do little more than search for occurrences of words. Thus ironically, we have the two extremes of rigidly structured and totally unstructured information—but very little in between.

CORRELATION WITH INFORMATION CONCEPTS

For information modeling purposes, one has to account for such concepts as entities and entity types, relationships and attributes, and naming. These are discussed in the following sections.

ENTITIES AND TYPES

There is a natural inclination to identify entities with records, since a record has the sense of being an integral object. It is an elementary unit of creation and destruction, as well as of data transmission, and records are classified into types just as entities are. Such a correspondence between entities and records would be enormously simplifying, giving us information modeling as a free by-product of data management technology.

Arguments against such a modeling approach are hampered by the lack of a good operational definition of the term "entity." But we can suggest some difficulties in reconciling record structures with certain "intuitively obvious" characteristics of entities. Thereafter we can either conclude that records have limited value for information modeling, or else adjust our intuitions about entities in order to get a better fit with record concepts.

The questions which might be asked to test the hypothesis that records represent entities are: How well do their characteristics match? Is there a 1:1 correspondence between them?

A Record Does Not Have All the Facts

Many facts have the form of a relationship between two entities (e.g., a department and an employee). Although it concerns both entities, such a fact is not likely to be replicated in the records representing both entities. At most, it will usually be included in the record of only one of the entities involved.

Quite often the fact will not occur in the record corresponding to either one. If the fact is a many-to-many relationship, such as employees and their skills, then normalized record systems do not permit the necessary repeating field to occur in either the employee record or the skill record. Normalized systems constrain a record to be a collection of single-valued facts. If a class has one instructor, then that can be mentioned in the class record, but not if there might be several instructors.

Thus a record cannot be characterized as containing "all the facts" about an entity.

Entities Are Not Always Single-Typed

If we intend to use a record to represent a real world entity, there is some difficulty in equating record types with entity types. It seems reasonable to view a certain person as a single entity (for whom we might wish to have a single record in an integrated database). But such an entity might be an instance of several entity types, such as person, employee, dependent, customer, stockholder, taxpayer, parent, instructor, student, mammal, physical object, property (slaves?), etc. it is difficult, within the current record processing technologies, to define a record type corresponding to each of these, and then permit a single record to simultaneously be an occurrence of several of the record types.

Note that we are not dealing with a simple nesting of types and subtypes: all employees are people, but some customers and stockholders are not. Nor are subtypes mutually exclusive: some people are employees, some are stockholders, and some are both.

In order to fit comfortably into a record-based discipline, we need to perceive entity types as though they did not overlap. We should perhaps think of customers and employees as always distinct entities, sometimes related by an "is the same person" relationship. But we then have to make arbitrary decisions about the placement of common information such as addresses and birthdates. Furthermore, one has to be very careful about the number of entities being modeled. If an employee is a stockholder, there will be two records for him; is he two entities? If a committee has five employees and five stockholders, how big is the committee?

(Bachman and Daya [4] and Smith and Smith [40] propose models in which multiple records can represent the multiple types, or roles, of an entity.)

Type Is Not Always Homogeneous

Even within a single type, there may be facts and naming conventions which are relevant to some occurrences and not others. These points were covered in the earlier discussion of horizontal homogeneity.

Entities Without Records

Most of the things mentioned in a database do not have any distinct records to represent them. These are the things we treat as attributes of other things, such as salaries, colors, birthdates, birthplaces, employers, spouses, addresses, etc. (While such things may be mentioned in multiple records, I do not think we can say they are "represented" by any one record.) Unfortunately, apart from the listing of examples, it is difficult to identify precise criteria for deciding whether something is an entity, and whether it is to be represented by a record.

In a normalized system, an entity might also fail to be represented by a single record if there did not happen to be any single-valued information about the entity. Suppose one had in mind to treat projects as entities, but all the information to be maintained about them turned out to be multivalued (in relational terms, we find no functional dependences on projects). That is, our projects can have multiple managers, multiple objectives, multiple start and stop dates, multiple budgets, and so on. Each such fact needs to be maintained in a distinct intersection record, and there might be no motivation to define a single record type or relation to represent the projects themselves. One would have record types (relations) called "project-manager," "project-objective," "project-dates," and so on, but none called simply "project."

Entities With Many Records

We might have too many records. As mentioned earlier, a common solution to the problem of overlapping types (e.g., employees and stockholders) is to define them as disjoint types, and allowing an entity (person) to be represented by a record in each type.

The "generalization" approach of Smith and Smith [40] yields multiple records per entity; it is not clear that any one of them could be said to "represent" the entity. The approach of Bachman and Daya [4] is similar in this respect, but they do postulate one underlying record (never materialized) per entity.

More generally, there is no discipline preventing the definition of several record types corresponding to one entity type. That is, we could have several record types defined over the same key, with each record type containing different attributes of the subject entity. One might be tempted to do this for economic reasons, e.g., to group together attributes which tend to be accessed together, or to physically segregate rarely used data. Regardless of the motivation, such a configuration is permitted in all record-based systems. Thus none of these systems really has a well-defined semantic establishing a 1:1 correspondence between entities and records.

Records Without Entities

Normalized systems require many-to-many facts to be represented in distinct record types of their own (so-called intersection records). Employee-skill records are a good example. What entity does one of these records represent? Not the employee, nor the skill. If it represents anything at all, the record represents the *relationship* between the employee and the skill. This might suggest the principle that relationships are entities, and ought in general to be represented by records. But some relationships are not represented by records, e.g., the relationship between an employee and his department. (That relationship is recorded in an employee record, but not represented by a distinct record of its own.) Obviously, it is only the many-to-many relationships which must be represented by distinct records (in a normalized system); are they the only ones which are entities?

There are three ways to take a consistent view of this situation:

(1) All relationships are entities, and some records represent multiple entities (as the same record "represents" both an employee and his relationship to a department).
(2) Relationships are not entities, and intersection records do not represent entities.
(3) Some relationships are entities and some are not depending on whether or not they are represented by intersection records.

It is a matter of judgment as to whether any of these views is acceptable.

Depending on what definitions we like, some intersection records might not even represent relationships. You might wish to consider the color of a car to be an attribute, and not a relationship. But if cars are multicolored, then their colors must be split out into separate car/color intersection records. Does the attribute now become a relationship? Is it an entity? What entity does the record represent?

If we do not care to think of such multivalued attributes as being entities in themselves, then we again have records which do not represent entities.

Records With Many Entities

If there is a 1:1 correspondence between certain entities, then a single record might be perceived as "representing" all of them. Employees and spouses provide an example, in a monogamous society. Since each spouse occurs in exactly one employee record, one could view these records as representing spouses just as well as employees. The perception is even more plausible if the spouses are uniquely identified (as they might be, by social security number), and if they occurred in every record (if, perhaps, company policy required all employees to be married).

Summary

"Entity" is not very well defined, for our purposes. To be absolutely fair, we should only conclude that record structures do not correspond to everyone's intuitions about the characteristics of entities. But it is quite difficult to establish a definition of "entity" which puts it in 1:1 correspondence with a normalized record, unless one starts with that as the definition.

RELATIONSHIPS

One Concept, Many Representations

A binary relationship is a fairly simple concept: a named link between two entities. But there are about a half dozen ways to implement binary relationships in record structures. (Schmid and Swenson [31] make a similar analysis.)

Most of these ways to implement binary relationships involve pairing identifiers of the two entities in one record. It might be in the record representing one entity or the other. It might be in a separate record (intersection record) representing the relationship itself. And it might be embedded in a record representing some other entity altogether (an employee record may include a relationship between the employee's spouse and the spouse's employer).

These alternatives correspond to several combinations in which the two entity identifiers might occur as keys in a record. One or the other might be the key, or they might together constitute the key, or neither might be in the key.

(Actually, there are other possibilities as well. One identifier or the other might be a subset of the key, which probably violates third normal form. Or they might together constitute a subset of the key, in which case they might be part of a compound name—to be discussed later. Or they might each constitute a key for the record, if there was a 1:1 relationship between them.)

In addition, a relationship might be represented indirectly, being implied by other relationships. If projects are assigned to single departments, and if each employee works on all of his department's projects, then the way to discover if an employee works on a certain project is to match the department numbers in the employee and project records. That is, an employee's assignment to a department and a project's assignment to the department together imply the employee's working on the project.

In record management systems which provide file structure in addition to record structure, even more options become available. Relationships might be represented by file order, or by the linkage of records into hierarchies or (CODASYL) sets.

For information modeling, the problem is what to do with this plethora of options [12, 26]. Why is it necessary to make such choices? What are the criteria? Do the criteria have anything to do with the semantics of the information, as distinguished from the economics of storing or processing the data? Do all users have to know which options have been chosen, and to adapt their processing accordingly?

Normalized systems reduce the number of options for one-to-many and many-to-many relationships, and generally force them to be treated differently from each other. (But not necessarily; one-to-many relationships can be represented in separate intersection records, though they hardly ever are.) Some differences force the information modeler to prematurely impose processing techniques on end users: one-to-many relationships can often be altered by updating field values, while many-to-many relationships in intersection records can only be altered by deleting and inserting records.

Relationships Are Not Described

The various representations available for relationships are often used for other purposes as well. Thus record descriptions rarely provide clear evidence of the presence of relationships—neither explicitly nor implicitly. A record with multi-field keys might represent a relationship, or the keys might constitute a qualified name for an entity. If neither field is in the key (e.g., spouse and spouse's employer), there is no mention at all of the relationship; the two fields appear to be independent facts about the employee.

There is no regular way to reflect the name of the relationship in the file description. Sometimes it is a record type name (intersection records), sometimes it is a field name (or a part of one), and sometimes it does not occur at all (e.g., implied relationships, or joins). When file structure is available, the relationship name might also be a (CODASYL) set name or, again, it might not occur at all if represented by file order or hierarchical structure.

And, when the relationship names do occur as field names, there is no discipline. There is rarely any clue in the field name (or even in a relational domain name, if provided) as to the record type representing the related entity. Consider a generalized query processor asked to find the name of the manager of a certain department. The department record probably has a field named MANAGER (perhaps with a domain named EMPLOYEE NUMBER). What tells the processor to look into a record type named EMP-RECS to find the name of the employee whose number occurs in that field?

Sometimes the field name combines the relationship and the entity type ("assigned-dept").

The same field name might signify the same or different relationships in different records, and different names might be used for the same relationship in different records. Since field names cannot be duplicated within a record, a relationship occurring more than once in a given record necessarily has different names (in a credit application, "employer" and "spouse's-employer" refer to occurrences of the same relationship).

In a good model for relationships, one might expect some direct way to declare relationships, specifying a name and some characteristics, without having to choose among a variety of (ambiguous) representational alternatives.

Attributes

There does not seem to be an effective way to characterize "attributes," or to distinguish them from relationships. Ironically, the most dominant correlate seems to be with record structures: if a field value is the key of some other record, then it represents a relationship; otherwise it is an attribute. This need to map things into recordlike terms seems to be the main force motivating a distinction between attributes and relationships. If we did not have a record-based implementation in mind, the distinction might go away [35].

NAMING, SYMBOLIC REFERENCE

In a pure record structure, most facts (relationships, attributes) are represented by including in one record symbolic identifiers of two or more things (e.g., employee number and department number, or employee number and salary) [32]. Such symbolic reference admits references to nonexistent entities (entities whose corresponding records are missing). Symbolic reference forces a strong interaction between the concepts of identifier and entity type, and encounters problems with synonyms and with changeable names.

When simple labels are not conveniently available, the record model permits arbitrary combinations of facts to be specified as identifiers capable of distinguishing among entity occurrences. This, in turn, leads to further problems, unless a number of constraints on the selected facts are carefully observed. Furthermore, in using such identifiers to refer to entities, multiple fields serve the function of a single field—generating ambiguities in the structure of the information represented by the record.

Simple Identifiers

To the extent that a record represents an entity (i.e., signifies its existence), symbolic reference permits references to nonexistent entities. A department number can occur in an employee record even if no corresponding department record exists. At best, there might be some check that the field contains a "plausible" department name, in terms of its syntax: the right number of the right kinds of characters. (In the proposals of Schmid and Swenson [31] and Smith and Smith [39] such existence dependences can be expressed and maintained.)

When arbitrary identifiers are assigned, such as employee numbers, then there is little question of uniqueness of identifier. But when some fact about the entity is given double duty as an identifying label for that entity, one has to be quite careful that the fact does, in fact, uniquely determine the entity. Presidential elections can be identified by the year in which they occur—provided we are absolutely certain there is no possible circumstance under which another presidential election might be held in the same year.

Symbolic identifiers rarely provide absolute identification of entities. At best, the identifiers are unique within entity type. One cannot tell which entity is being referenced just by examining a field value; one has to have supplementary knowledge as to which entity type is involved. Nothing in the data tells us whether 123456789 identifies a person or a machine. Such information is almost never included with the data, nor with the record description. That is, a field description rarely specifies the entity type, or record type, whose keys will occur in that field. The domain construct of the relational model provides limited assistance in this area, as mentioned in the earlier discussion of field names.

In a pure record structure (one with no pointers or other file structure interconnecting the records), the only way to detect a reference to the same thing in several records is to find a match in some corresponding field values. (For example, that is the basis of the relational join.) Synonyms and aliases interfere with this process. At best, one could execute a chain of path-following operations, if one knew which record types provided which synonym linkages. That is the only way to detect that, e.g., a social security number in one record referred to the same person as an employee number in another record.

There are often several *kinds* of identifiers by which an entity can be identified uniquely, such as employee numbers and social security numbers. One now has to know not only which entity type is being referenced in the field, but also which kind of identifier is being used. And one has to know which other record type to go to in order to "translate" one name type into the other.

In an obvious way, the extent of an entity type affects the choice of identifier. Facts which are unique over a small set of entities may not be unique over a larger set.

Presidential elections in one country can be identified simply by the year of occurrence. But if the entity type were perceived as a larger set of elections, then the identification would have to include additional facts, such as the office (governor, mayor, etc.) and political unit involved (name of the country, state, city, etc.).

In a converse way, identifiers can affect the perception of entity types. In order to provide record keys, it is often necessary to arbitrarily choose one kind of identifier as a "primary" identifier for an entity. Two constraints then impose boundaries on entity types: a key value is meaningful within exactly one record type, and each record of the type must contain a key value. That is, key values must be in 1:1 correspondence with the set of entities represented, and the tendency is to think of a set as an entity type if it corresponds to the scope of a unique identifier. Unfortunately, different kinds of identifiers may have overlapping but unequal scopes, leading to conflicting choices of entity types. Employee numbers, social security numbers, military service numbers, etc., each identify different sets of people, each leading to a different concept of the "entity type" involved. When identified by ISBN's, books constitute a different entity type from films. When identified by Library of Congress numbers, they are all publications. Thus in general, the varying scopes of identifiers can have an excessive and potentially confusing influence on the establishment of entity types. If names were not tied so intimately to types, it might be possible to deal with types more naturally.

The tight coupling between types and names forces vertical homogeneity, i.e., the inability to represent facts which involve multiple entity types. The name formats are likely to be different between types; when the formats are the same, names might not be unique across types.

Sometimes an entity can have multiple identifiers of the *same* kind. A person can have several social security numbers, and a book might have several ISBN's. Names and addresses can be spelled and abbreviated in a variety of ways. Again, it becomes harder to detect references to the same entity when different identifiers are used. (Consider the difficulty of purging duplicates out of a mailing list.) Furthermore, these identifiers cannot all occur in the record which represents the entity (assuming normalized records), and they certainly cannot all be keys of the record. At most, one of them has to be arbitrarily selected as the primary identifier. For the others, a new intersection record type has to be defined, with each such record "translating" one secondary identifier into a primary identifier. Now, if some record contains a secondary identifier, it is necessary to know that the field is not a key into the entity record type, but must first be translated via the intersection records. If the field might contain either primary or secondary identifiers, then the retrieval algorithm is even more complex. (And, incidentally, one might ask what entities are being represented by the intersection records. They are a distinct record type from the one representing the entities. Their keys are the secondary identifiers, with a separate record for each one, hence several records per entity.)

Sometimes the identifier chosen for an entity is not, strictly speaking, the name of that entity, but of a related entity. This practice can confuse the underlying semantic structure of the information. Many facts are available, for example, about elections. One might reasonably expect to ask, in a symmetric way, who won Election No. 10 and when Election No. 10 was held. But if elections are named by their dates, e.g., the "1948 election," it suddenly becomes absurd to ask when it was held. Strictly speaking, we are dealing with an unnamed entity being identified by means of a related entity, but the record descriptions give no hint of this structure. (If it disturbs you to think of a year as being a related entity, then imagine that presidents could only be elected once, and we therefore named elections by their winners. Then it becomes equally absurd to ask who won the "Truman election," although it is meaningful to ask who lost it, and when it occurred.)

Another way to see the semantic problem is in terms of functional dependences. On an entity level, an election uniquely determines its date and its winner, and hence dates and winners ought to be functionally dependent on elections. But if the relation representing this information contains only a date column and a winner column, then there is no way to express these two functional dependences. In effect, the existence of three distinct entities is not acknowledged in any way. This is one respect in which relations and functional dependences do not mirror relationships among entities.

And, finally, it is not sufficient for a fact to uniquely identify an entity. In order to be useful in a record structure, it had better be an unchangeable fact. It is highly undesirable to have to change that fact in every place where the entity is referenced. Thus while a phone number might uniquely determine an office at any given moment, it is not a good way to identify offices if the phones tend to be moved around.

Compound Identifiers

All of these concerns—and more—apply when compound facts are used to identify entities in records.

We use for an example the identification of dependents of employees, perhaps for a benefits database. Names of dependents are certainly not unique, but we might assume that no employee has two dependents with the same name. Then a dependent could be identified by the combination of his name and the related employee's identifier (as in [8]). (We will refer to that related employee as the "sponsor.")

To begin with, the concerns about uniqueness, existence, synonyms, scope, and changeability must be examined more carefully, because there are more facts involved.

We are depending on the uniqueness of names of one employee's dependents, which might be a questionable assumption if the employee has adopted children, or has remarried, or if the employee's parents, grandchildren, siblings, etc. also qualified as dependents in addition to his children. More simply, his child and spouse might have the same name.

In references to the dependent from other records we have, as before, no assurance that the dependent record itself exists. But a second level of existence dependence is now introduced. In order for the dependent's identification to be meaningful, there must be some assurance that the sponsor's employee record also exists.

Synonyms can easily arise, if a dependent was related to more than one employee. The dependent has as many valid identifiers as he has sponsors. A compound identifier thus need not even be restricted to a many-to-one relationship. If all we are after is the ability to select dependents, then all we need is that no employee have two dependents with the same name. The relationship could be many-to-many.

This gives us a one-way uniqueness. A qualified name identifies exactly one dependent, but we cannot tell whether two qualified names might refer to the same dependent. If two employees had a dependent named "Joe," what indicates whether it is the same dependent? Special pains must be taken to detect that the dependents of several employees might be the same person, in order to properly coordinate benefits. A separate "translation" record type is now needed to establish the equivalences between identifiers. (And the synonym problem is compounded if the sponsors themselves have synonyms.)

It is possible to designate one identifier (involving one sponsor) as the "primary" identifier of the dependent. But it may be necessary in some situations to permit any identifier to be used, requiring a search of both the translation records and the dependent records. Furthermore, this leads to a potential violation of the constancy requirement: a dependent's primary sponsor will change if that sponsor leaves the company but the dependent is still related to other employees. Then all references to that dependent have to be modified. In fact, it is probably not a good idea to use employees to qualify dependent names in an implementation which forbids modification of record keys.

Scope of identifiers is, of course, still a crucial concern. The facts chosen for the compound name must be relevant and known for all occurrences of the entity type being identified. For example, if the company charitably expanded its benefits program to provide aid to all needy people in the community, then some recipients would not have any related employee to use in their identifiers. Some other form of identification would have to be devised.

So far, all these problems of compound names have just been extensions of similar problems which existed for simple names. A whole new set of problems derives from the fact that compound names force an entity to be referenced by multiple fields in a record.

There is a three-way ambiguity. Multiple fields in a record are used to represent three distinct semantic constructs: the compound name of an entity, a relationship among entities, or multiple independent facts about an entity.

Spurious Joins. The resemblance between compound names and independent facts can lead to "phantom entities" and spurious joins. Whenever an employee number and a person's name occur in any two fields of a record, this could be construed as the compound identifier of a dependent. In particular, a relational join can be performed over such fields.

An employee's name and his manager's employee number, in one record, could be matched up with a customer's name and his salesman's employee number in another record, in the mistaken belief that such pairs of fields constituted qualified names for the same person. More simply, a dependent record might include another employee number field besides his sponsor (e.g., a benefits administrator); a join could be done on the wrong employee field, mistaking this person for a dependent of that other employee (who might not even have any dependents).

The domain concept in the relational model is supposed to control joinability. In relational theory, joins may only be performed on columns based on the same domains, presumably assuring that the same entity is occurring on both sides as a "link." The problem is that relations are fundamentally defined as aggregations of single columns, each column based on a domain. There is no concept of a domain encompassing multiple columns, although entities may be identified by multiple columns. The domain concept cannot be applied in such a way that two columns in a relation must always be treated as a unit, and may only be joined with other pairs of columns defined to constitute similar units. In the current example, the two fields might have separate domains specified as "employee number" and "person name"; there is no place to establish "dependents" as a domain. And even if the second domain was specified as "dependent name," it still could be paired with any other employee number in the same record.

Thus domains do not readily correspond to entity types, when the entities are identified by compound names.

Reducibility Dilemma. The resemblance between compound names and independent facts leads to a dilemma in the theory of reducible relations [15, 16, 19, 29].

Consider a person's birthday. On the face of it, this is an elementary (irreducible) fact—a simple binary relationship between a person and a certain day in the past. And, if we happen to represent dates in Julian notation (one field), then birthday actually has the structure of an elementary fact. But if we choose to change the *naming* of the date to the more conventional notation involving three fields, then we have a record containing four fields. This record can now be reduced to three binary records: person and year, person and month; and person and day of month. The original birthday record can always be recovered by joining these three.

The same analysis, and ambiguity, applies whenever a composite naming convention is selected for an entity. City of birth, for example, is an irreducible fact if globally unique city codes are used; it is reducible if the city is identified by the composite of, e.g., city, state, and country names.

Composite names are in general *not* precisely equivalent in function to simple unique identifiers for the same entities. Composite names almost always convey additional information; when used in lieu of simple names they necessarily change the underlying structure of the information. A simple name simply designates an entity; a composite name does that, but it simultaneously informs us about other related entities. A city code simply designates a city; the conventional notation may additionally tell us the state and country in which it is located. A Julian date simply designates a certain day (if we do not bother to do certain computations); the conventional notation additionally tells us the year and month in which it occurred, as well as the day of the month. In the role of designating a single entity, a compound name could be part of an irreducible fact; in the role of providing auxiliary information about related entities, it leads to reducibility.

Confusion of Subject. The resemblance between compound names and relationships confuses the subject matter of a record, namely the question of what entity is being represented by a record. A record having employee number and dependent name as a compound key might be a dependent record (representing a dependent), or it might be an intersection record representing the relationship between an employee and a dependent. In the first case, it would be appropriate to include the dependent's age in the record, since that is a fact about the dependent:

```
EMPLOYEE-NUM | DEPENDENT-NAME | AGE
.............. key ..............
```

In the second case, the record might specify the kinship between the dependent and the employee:

```
EMPLOYEE-NUM | DEPENDENT-NAME | KINSHIP
.............. key ..............
```

The first fact is really about the dependent alone, while the second is about the relationship between the dependent and an employee. But the structure of the two in "irreducible" form is indistinguishable. Thus if naming conventions are not separated from entity representation, "irreducible" records still do not model the structure of information unambiguously. (To see the significance, compare what happens to the preceding two structures if dependents were named simply, e.g., by social security numbers.)

In unreduced records, a composite key is likely to be serving both roles simultaneously. It would not be unusual to see the two records shown above combined into one (since they have the same key), containing both age (a fact about the dependent) and kinship (a fact about the relationship). It is thus ambiguous as to which entity is really represented by this record.

Ironically, although "intersection records" are commonly accepted as the construct which represents many-to-many relationships, it is, in fact, hard to know which records are serving as intersection records.

Surrogates

A precise model of information should distinguish carefully between the structure of entities being modeled and the various structures of names which might be associated with them. This implies a distinction in the model between entities and traditional data items.

Some alternative models suggest that some sort of an internal construct be used to represent an entity, acting as a "surrogate" for it [1, 19]. This surrogate would occur in data structures wherever the entity is referenced, and naming problems would at least be isolated by keeping structured or ambiguous identifiers off to one side, outside the structures representing attributes and relationships.

Since these surrogates must eventually be implemented inside the computer in some form of symbol string, it is sometimes held that such surrogates are themselves nothing but symbolic identifiers.

It is useful to be aware of some fundamental differences between surrogates and ordinary identifiers:

(1) A surrogate might not be exposed to users. Only ordinary identifiers need pass between user and system. In concept, models involving surrogates can behave as though a fact (e.g., the assignment of an employee to a department) was treated in two stages. First, the surrogates corresponding to the employee and department identifiers are located (i.e., name resolution). Then the two surrogates are placed in association with each other, to represent the fact.

(2) Users do not specify the format, syntax, structure, uniqueness rules, etc. for surrogates.

(3) Surrogates are globally unique, and have the same format for all entities. The system does not have to know the entity type (or the identifier type) before knowing which entity is being referenced, or before knowing what the surrogate format will be.

(4) Surrogates are purely information-free. They do not imply anything about any related entities, nor any kind of meaningful ordering.

SEMANTIC STRUCTURE AND THIRD NORMAL FORM

In the absence of any additional discipline, it is difficult to guess the semantic structure implied by a record format. A given field might include multiple facts by encoding, or with internal punctuation. A given fact might occur in any of several fields: in one of our example vehicle assignment records, the assignee would occur in the third field of some records and the fourth field of other records. A given field might give information about different things in different records: a clever programmer might design a single "maiden name" field to refer either to an employee or to the employee's spouse, depending on which was female. In general, the possibility of tricks such as these obscures the underlying semantic structure.

The semantic structure suggested by a relation under the discipline of third normal form [11, 20] is one in which each record represents a single "subject" entity (identified by the key), with the other fields being "direct" facts about the subject entity. That is, the nonkey fields are functionally dependent on the key, and there are no other functional dependences.

There are some difficulties here also. In the first place, the role of functional dependences in relational theory is unclear, at least as reflected in implementations. The assumption tends to be that functional dependences (if specified at all) have been used during the design phase of the database to insure that relations are in third normal form, and then discarded. They do not seem to be present at run time to explain the semantic structure of the data.

Secondly, functional dependences deal only with 1:*n* relationships and not *m*:*n* relationships. However, some recent work [14] offers extensions into this area.

Thirdly, functional dependences merely specify dependences without naming the relationships involved. Thus, e.g., it cannot be determined whether several functional dependences (between different pairs of columns) are based on the same relationship. Consider, for example, a credit application record containing the applicant, employer, spouse, and spouse's employer. Functional dependences give no clue that the same relationship exists between the first and second columns as between the third and fourth.

Furthermore, third normal form allows several keys to remain in the same relation, even if they identify different entities. This can occur when entities are in a 1:1 relationship, as with employees and spouses. One of the keys is typically selected as "primary," which can suggest that only one entity (e.g., the employee) is the subject of the record. If functional dependences are discarded after the database design phase, there is nothing left to dispel this illusion. The remaining structure implies, for example, that "spouse's birthdate" is not an attribute of the spouse, but only of the employee. The structure is vulnerable to update anomalies, with the implication that the "spouse" and "spouse's birthdate" attributes of an employee may be updated independently of each other.

Still another concern is that it may not be possible to model the *attributes* of 1:*n* and 1:1 relationships, because only "full" functional dependences are considered in the determination of third normal form. A full functional dependence is based on the "smallest" possible subject which determines the related fact. Thus while the relationship between an employee and his department may have an assignment-date attribute, that date is not "fully" dependent on both employee and department. There is only one (current) assignment date for each employee, and hence that date has a full functional dependence on the employee alone. (That is, knowing the employee is sufficient to determine the date.) To the extent that only full dependences are considered, functional dependences distort the semantic structure of information. Attributes of 1:*n* and 1:1 relationships are always transformed into attributes of one of the related entities. Only the attributes of *m*:*n* relationships are preserved in functional dependences.

Finally, the most serious concern is that functional dependences are expressed on the name level rather than the entity level. One consequence is the failure to model the existence of entities whose names are given to other entities (as in the earlier example of elections named via years or winners).

Another consequence is that dependences are only expressed when the subjects are uniquely named. In the example of employees and spouses, functional dependences on the spouses would only be expressed if the spouses were uniquely named (as by social security number). In the more common case, no functional dependences could be written, since the same spouse *name* might occur in different records with different birthdates, employers, etc. Spouses would not in any way be designated as the subjects of any information, or as candidate keys.

(We could uniquely identify spouses by the combination of spouse name and employee number, as a qualified name. But functional dependences based on this pair of fields would not be full dependences, since the attributes—such as "spouse's birthdate"—are uniquely determined by employee number alone. As mentioned earlier, such nonfull dependences are discarded in the determination of third normal form.)

Because of the dependence on naming, a large number of relationships in a typical database fail to be reflected in functional dependences, and hence fail to be subject to the discipline of third normal form. In a personnel file, attributes of such things as previous employers, banks, schools, relatives, etc. tend to be duplicated by embedding in multiple employee records. The functional dependence between a school and its address could not be specified, because several schools might have the same name. Hence, if several employees attended the same school, the address of that school might be included in each employee's record, without violating third normal form. All such cases are exposed to the update anomalies which third normal form is supposed to prevent.

A special case of this problem is not only tolerated but in fact encouraged by the relational model. Compound keys, in which the name of an entity includes the name of a related entity, plainly constitute a replication of information, thereby violating the spirit if not the letter of third normal form.

Consider an earlier example, where departments were uniquely named only within divisions, so that company records had to use a compound of department name qualified by division name. On the entity level, a department uniquely determines its division (a given department is, after all, in only one division). But on the name level, a department name does *not* uniquely determine the corresponding division—that is precisely why qualification is needed in the first place. Hence no functional dependence can be written here, even though there is a many-to-one relationship between the entities involved.

A department's division is mentioned in every record which references that department. If the division of a department changes, that single fact must be changed wherever that department is mentioned—precisely what third normal form should avoid.

There is a pragmatic solution. The update problem is best dealt with by legislating it away: forbid the update of keys. This is common in most implementations, though not a formal part of relational theory.

Third normal form is very sensitive to naming conventions. Giving departments globally unique identifiers would represent no real change in the semantic structure of the information. But a functional dependence could now be written between departments and divisions, and the records in question would no longer be in third normal form.

To conclude: records in third normal form can include a great many relationships which are not suggested at all in the record descriptions, even in the form of functional dependences.

CONCLUSIONS

We have outlined a number of ways in which record structures fail to model the semantics of information accurately and unambiguously. Other models deal with these problems, with varying degrees of success. A discussion of such alternatives is generally beyond the scope of this paper.

Just briefly, we can observe that file structures (hierarchies, CODASYL sets) overcome some functional limitations, but by increasing rather than decreasing the variety of representational alternatives. Bachman and Daya [4] and Smith and Smith [40] address some of the problems of types, but still in a record-based framework. Binary relation (or "elementary fact") models are generally more successful in coping with the modeling problems, though none have done so completely. Such models are more directly based on semantic concepts, .e.g., entities and the network of relationships among them, rather than on recordlike structures. The model described by Biller and Neuhold [5] is excellent in this respect. Other models along these lines include the works (referenced below) of Abrial, Bracchi, Falkenberg, Griffith, Hall, Roussopoulos, Schmid, Senko, and Sowa.

ACKNOWLEDGMENTS

Many people made valuable comments on earlier versions of this paper, including Chris Date, Bob Engles, Bob Griffith, Roger Holliday, Lucy Lee, and especially Paula Newman, who suggested some of the ideas. I am also deeply indebted to the referees, especially Mike Senko, who instigated major improvements in the paper.

REFERENCES

1. ABRIAL, J.R. Data semantics. In *Data Base Management,* J. W. Klimbie and K. L. Koffeman, Eds., North-Holland Pub. Co., Amsterdam, 1974.
2. ANSI/X3/SPARC. Study Group on Data Base Management Systems. Interim Rep., Feb. 1975; also FDT (Bulletin of ACM SIGMOD) *7,* 2 (1975).
3. The ANSI/X3/SPARC DBMS Framework. Report of the Study Group on Data Base Management Systems, D. Tsichritzis and A. Klug, Eds., AFIPS Press, Montvale, N.J., 1977.
4. BACHMAN, C.W., AND DAYA, M. The role concept in data models. Proc. Third Int. Conf. Very Large Data Bases, Database (ACM) *9,* 2 (Fall 1977), SIGMOD Record (ACM) *9,* 4 (Oct. 1977), 464-476.
5. BILLER, H., AND NEUHOLD, E.J. Semantics of data bases: The semantics of data models. *Inform. Syst. 3,* 1 (1978).
6. BRACCHI, G., PAOLINI, P., AND PELAGATTI, G. Binary logical associations in data modeling. In *Modelling in Data Base Management Systems,* G. M. Nijssen, Ed., North-Holland Pub. Co., Amsterdam, 1976.
7. CHAMBERLIN, D.D. Relational data base management systems. *ACM Comptng. Surveys 8,* 1 (March 1976), 43-66.

8. CHEN, P.P.S. The entity-relationship model: Toward a unified view of data. *ACM Trans. Database Syst. 1,* 1 (March 1976), 9-36.
9. CODD, E.F. A relational model of data for large shared data banks. *Comm. ACM 13,* 6 (June 1970), 377-387.
10. CODD, E.F. Normalized data base structure: A brief tutorial. ACM SIGFIDET Workshop on Data Description, Access, and Control, San Diego, Nov. 1971, pp. 1-18.
11. CODD, E.F. Further normalization of the data base relational model. In *Data Base Systems* Courant Computer Science Symposia 6, R. Rustin, Ed., Prentice-Hall, Englewood Cliffs, N.J., 1972; also IBM Res. Rep. RJ909.
12. CODD, E.F., AND DATE, C.J. Interactive support for non-programmers: The relational and network approaches. ACM SIGMOD Workshop on Data Description, Access, and Control (Vol. 2), May 1974, pp. 11-41; also IBM Res. Rep. RJ1400.
13. DATE, C.J. *An Introduction to Database Systems*, Addison-Wesley, Reading, Mass., Second ed., 1977.
14. FAGIN, R. Multivalued dependencies and a new normal form for relational databases. *ACM Trans. Database Syst. 2,* 3 (Sept. 1977), 262-278; also IMB Res. Rep. RJ1812.
15. FALKENBERG, E. Concepts for modelling information. In *Modelling in Data Base Management Systems,* G. M. Nijssen, Ed., North-Holland Pub. Co., Amsterdam, 1976.
16. FALKENBERG, E. Significations: The key to unify data base management. *Inform. Syst. 2,* 1 (1976), 19-28.
17. GRIFFITH, R.L. Information structures. IBM Tech. Rep. TR03.013, IBM, San Jose, Calif., May 1976.
18. Data Base Management System Requirements. Joint Guide-Share Data Base Requirements Group, Nov. 1970.
19. HALL, P.A.V., OWLETT, J., AND TODD, S.J.P. Relations and entities. In *Modelling in Data Base Management Systems,* G.M. Nijssen, Ed., North-Holland Pub. Co., Amsterdam, 1976.
20. KENT, W. A primer of normal forms. Tech. Rep. TR02.600, IBM, San Jose, Calif., Dec. 1973.
21. KENT, W. New criteria for the conceptual model. In *Systems for Large Data Bases,* P. C. Lockemann and E. J. Neuhold, Eds., North-Holland Pub. Co., Amsterdam, 1977.
22. KENT, W. Entities and relationships in information. In *Architecture and Models in Data Base Management Systems,* G.M. Nijssen, Ed., North-Holland Pub. Co., Amsterdam, 1977.
23. KERSCHBERG, L., OZKARAHAN, E.A., AND PACHECO, J.E.S. A synthetic English query language for a relational associative processor. Proc. Second Int. Conf. Software Eng., San Francisco, 1976, pp. 505-519.
24. KLIMBIE, J.W., AND KOFFEMAN, K.L., Eds. *Data Base Management,* North-Holland Pub. Co., Amsterdam, 1974. (Proc. IFIP Working Conf. Data Base Management, Cargese, Corsica, France, April 1974.)
25. LOCKEMANN, P.C., AND NEUHOLD, E.J., Eds. *Systems for Large Data Bases,* North-Holland Pub. Co., Amsterdam, 1977. (Proc. Second Int. Conf. Very Large Data Bases, Sept. 1976, Brussels.)
26. NIJSSEN, G.M. Two major flaws in the CODASYL DDL 1973 and proposed corrections. *Inform. Syst. 1* (1975), 115-132.
27. NIJSSEN, G.M. *Modelling in Data Base Management Systems,* North-Holland Pub. Co., Amsterdam, 1976. (Proc. IFIP TC-2 Working Conf., Freudenstadt, W. Germany, Jan. 1976.)
28. NIJSSEN, G.M. *Architecture and Models in Data Base Management Systems,* North-Holland Pub. Co., Amsterdam, 1977. (Proc. IFIP TC-2 Working Conf., Nice, France, Jan. 1977.)
29. RISSANEN, J., AND DELOBEL, C. Decomposition of files, a basis for data storage and retrieval. IBM Res. Rep. RJ1220, IBM Res. Lab., San Jose, Calif., May 1973.
30. ROUSSOPOULOS, N., AND MYLOPOULUS, J. Using semantic networks for data base management. Proc. Int. Conf. Very Large Data Bases, 1975, pp. 144-172. (available from ACM, New York)
31. SCHMID, H.A., AND SWENSON, J.R. On the semantics of the relational model. Proc. ACM SIGMOD Int. Conf. Manage. of Data, 1975, pp. 211-223.
32. SENKO, M.E., ALTMAN, E.B., ASTRAHAN, M.M., AND FEHDER, P.L. Data structures and accessing in data base systems. *IBM Syst. J. 12* (1973), 30-93.
33. SENKO, M.E. The DDL in the context of a multilevel structured description: DIAM II with FORAL. In *Data Base Description,* B.C.M. Douque and G. M. Nijssen, Eds., North-Holland Pub. Co., Amsterdam, 1975, pp. 239-257; also IBM Res. Rep. RC5073.
34. SENKO, M.E. Information systems: Records, relations, sets, entities, and things. *Inform. Syst. 1,*

1 (1975), 1-13.
35. SENKO, M.E. DIAM as a detailed example of the ANSI SPARC architecture. In *Modelling in Data Base Management Systems,* G. M. Nijssen, Ed., North-Holland Pub. Co., Amsterdam, 1976.
36. SHARMAN, G.C.H. A new model of relational data base and high level languages. Tech. Rep. TR. 12.136, IBM United Kingdom, Feb. 1975.
37. ACM SIGFIDET Workshop on Data Description, Access, and Control, Nov. 1971, San Diego, Calif., E. F. Codd and A. L. Dean, Eds.
38. ACM SIGMOD International Conference on Management of Data, May 1975, San Jose, Calif., W. F. King, Ed.
39. SMITH, J.M., AND SMITH, D.C.P. Database abstractions: Aggregation. *Comm. ACM 20,* 6 (June 1977), 405-413.
40. SMITH, J.M., AND SMITH, D.C.P. Database abstractions: Aggregation and generalization. *ACM Trans. Database Syst. 2,* 2 (June 1977), 105-133.
41. SOWA, J.F. Conceptual graphs for a data base interface. *IBM J. Res. and Develop. 20,* 4 (July 1976), 336-357.
42. TAYLOR, R.W., AND FRANK, R.L. CODASYL data base management systems. *ACM Comptng. Surveys 8,* 1 (March 1976), 67-104.
43. TSICHRITZIS, D., AND LOCHOVSKY, F.H. Hierarchical data base management systems. *ACM Comptng. Surveys 8,* 1 (March 1976), 105-124.
44. Proceedings of the International Conference on Very Large Data Bases, Framingham, Mass., 1975. (available from ACM, New York)
45. Proc. of the Second Int. Conf. Very Large Data Bases, Brussels, 1976.
46. Proceedings of the Third International Conference on Very Large Data Bases, Oct. 1977, Tokyo, Japan, Joint Issue Data Base *9* (ACM) 2 (Fall 1977), SIGMOD Record (ACM) *9,* 4 (Oct. 1977).

Received June 1977; revised February 1978 and June 1978